NASA Contractor Report 195360

Investigation of Advanced Counterrotation Blade Configuration Concepts for High Speed Turboprop Systems

Task 8–Cooling Flow/Heat Transfer Analysis User's Manual

E.J. Hall, D.A. Topp, N.J. Heidegger, and R.A. Delaney
Allison Engine Company
Indianapolis, Indiana

September 1994

Prepared for
Lewis Research Center
Under Contract NAS3−25270

National Aeronautics and
Space Administration

Contents

List of Figures

List of Tables

NOTATION

A list of the symbols used throughout this document and their definitions is provided below for convenience.

Roman Symbols

e . . . total internal energy
i . . . first grid index of numerical solution
j . . . second grid index of numerical solution
k . . . third grid index of numerical solution or thermal conductivity
n . . . rotational speed (revolutions per second) or time step level
r . . . radius or radial coordinate
t . . . time
u . . . velocity in the axial direction
v . . . velocity in the radial direction
w . . . velocity in the circumferential direction
x . . . Cartesian coordinate system coordinate
y . . . Cartesian coordinate system coordinate
z . . . Cartesian coordinate or cylindrical coordinate system axial coordinate
ADPAC . . . Advanced Ducted Propfan Analysis Codes
ADPOST . . . ADPAC post processing program
ASCII . . . American Standard Code for Information Interchange

CFL ... Courant-Freidrichs-Lewy number ($\Delta t/\Delta t_{max,stable}$)
$CHGRIDV2$ ... Ducted propfan grid generation code
D ... diameter (units of length)
$FULLPLOT$ ... PostScript x-y plotting program
J ... advance ratio ($J = U/nD$)
M ... Mach number
$MAKEADGRID$ ... ADPAC multiple-block mesh assembly program
$MULAC$ ... NASA-Lewis Compressor Mesh Generation Program
N ... Number of blades
R ... gas constant or residual or maximum radius
$ROTGRID$ ... ADPAC mesh rotation program
$SDBLIB$ ... Scientific DataBase Library (binary file I/O routines)
$SETUP$ ... ADPAC file setup program
$TIGG3D$ ... NASA-Lewis multiple splitter mesh generation program
U ... Freestream or flight velocity (units of length/time)
V ... volume

Greek Symbols

γ ... specific heat ratio
Δ ... calculation increment
ρ ... density
μ ... coefficient of viscosity

Superscripts

$[\ ]^n$ ... time level or iteration index

Subscripts

$[\ ]_{coarse}$ ... coarse mesh value
$[\ ]_{fine}$ ... fine mesh value
$[\ ]_{i,j,k}$ ... grid point index of variable
$[\ ]_{max}$ ... maximum value
$[\ ]_{min}$ ... minimum value
$[\ ]_{non-dimensional}$ ... non-dimensional value
$[\ ]_{ref}$ ... reference value
$[\ ]_{stable}$ ... value implied by linear stability
$[\ ]_{total}$ ... total (stagnation) value

Chapter 1

SUMMARY

The primary objective of this study was the development of a time-marching three-dimensional Euler/Navier-Stokes aerodynamic analysis tool to predict steady and unsteady compressible transonic flows about ducted and unducted propfan propulsion systems employing multiple blade rows. The current version of the computer codes resulting from this study are referred to as *ADPAC08* (Advanced Ducted Propfan Analysis Codes-Version 8). This report is intended to serve as a computer program user's manual for the *ADPAC08* code developed under Tasks VII and VIII of NASA Contract NAS3-25270.

The *ADPAC08* program is based on a flexible multiple-block grid discretization scheme permitting coupled 2-D/3-D mesh block solutions with application to a wide variety of geometries. For convenience, several standard mesh block structures are described for turbomachinery applications, although the code is by no means limited to these configurations. Aerodynamic calculations are based on a four-stage Runge-Kutta time-marching finite volume solution technique with added numerical dissipation. Steady flow predictions are accelerated by a multigrid procedure. The consolidated code generated during this study is capable of executing in either a serial or parallel computing mode from a single source code. Numerous examples are given in the form of test cases to demonstrate the utility of this approach for predicting the aerodynamics of modern turbomachinery configurations.

Chapter 2

INTRODUCTION

This document contains the Computer Program User's Manual for the consolidated *ADPAC08* (Advanced Ducted Propfan Analysis Codes - Version 8) Euler/Navier-Stokes analysis developed by the Allison Engine Company under Tasks VII and VIII of NASA Contract NAS3-25270. The objective of these studies was to develop a three-dimensional time-marching Euler/Navier-Stokes analysis for aerodynamic/heat transfer analysis of modern turbomachinery flow configurations. The analysis is capable of predicting both steady state and time-dependent flowfields using coupled 2-D/3-D solution concepts (described in detail in Section 1.3). The consolidated code was developed to be capable of either serial execution or parallel execution on massively parallel or workstation cluster computing platforms from a single source. The serial/parallel execution capability is determined at compilation. Throughout the rest of this document, the aerodynamic analysis is referred to as *ADPAC08* to signify that it is version 8 of the *ADPAC* series of codes.

A theoretical development of the *ADPAC08* program is outlined in the Final Report for Task V of NASA Contract NAS3-25270 [1]. Additional information will be presented in the Final Reports for Tasks VII and VIII of NASA Contract NAS3-25270 to be published in 1995 and 1994, respectively. In brief, the program utilizes a finite-volume, time-marching numerical procedure in conjunction with a flexible, coupled 2-D/3-D multiple grid block geometric representation to permit detailed aerodynamic simulations about complex configurations. The analysis has been tested and results verified for both turbomachinery and non-turbomachinery based applications. The ability to accurately predict the aerodynamics due to the interactions between adjacent blade rows of modern, high-speed turbomachinery was of particular interest during this program, and therefore, emphasis is given to these types of calculations throughout the remainder of this document. It should be emphasized at this point that although the *ADPAC08* program was developed to analyze the steady and unsteady aerodynamics of high-bypass ducted fans employing multiple blade rows, the code possesses many features which make it practical to compute a number of other complicated flow configurations as well.

2.1 Multiple-Block Solution Domain Concepts

In order to appreciate and utilize the features of the *ADPAC08* solution system, the concept of a multiple-block grid system must be fully understood. It is expected that the reader possesses at least some understanding of the concepts of computational fluid dynamics (CFD), so the use of a numerical grid to discretize a flow domain should not be foreign. Many CFD analyses rely on a single structured ordering of grid points upon which the numerical solution is performed (the authors are aware of a growing number of unstructured grid solution techniques as well, but resist the temptation to mention them in this discussion). Multiple-block grid systems are different only in that several structured grid systems are used in harmony to generate the numerical solution. This concept is illustrated graphically in two dimensions for the flow through a nozzle in Figures 2.1-2.3.

The grid system in Figure 2.1 employs a single structured ordering, resulting in a single computational space to contend with. The mesh system in Figure 2.2 is comprised of two, separate structured grid blocks, and consequently, the numerical solution consists of two unique computational domains. In theory, the nozzle flowpath could be subdivided into any number of domains employing structured grid blocks resulting in an identical number of computational domains to contend with, as shown in the 20 block decomposition illustrated in Figure 2.3. The complicating factor in this domain decomposition approach is that the numerical solution must provide a means for the isolated computational domains to communicate with each other in order to satisfy the conservation laws governing the desired aerodynamic solution. Hence, as the number of subdomains used to complete the

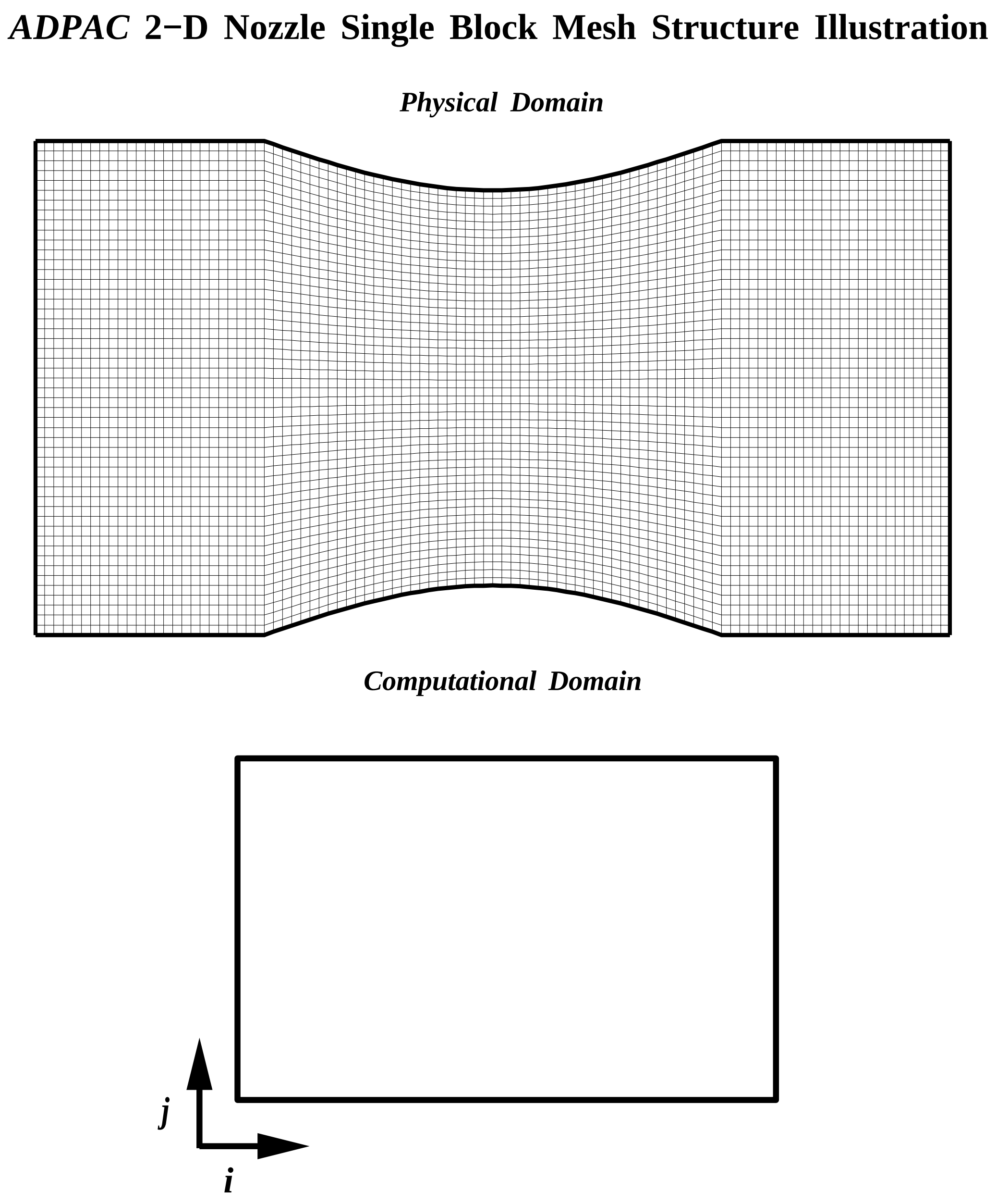


Figure 2.1: *ADPAC08* 2-D Single Block Mesh Structure Illustration

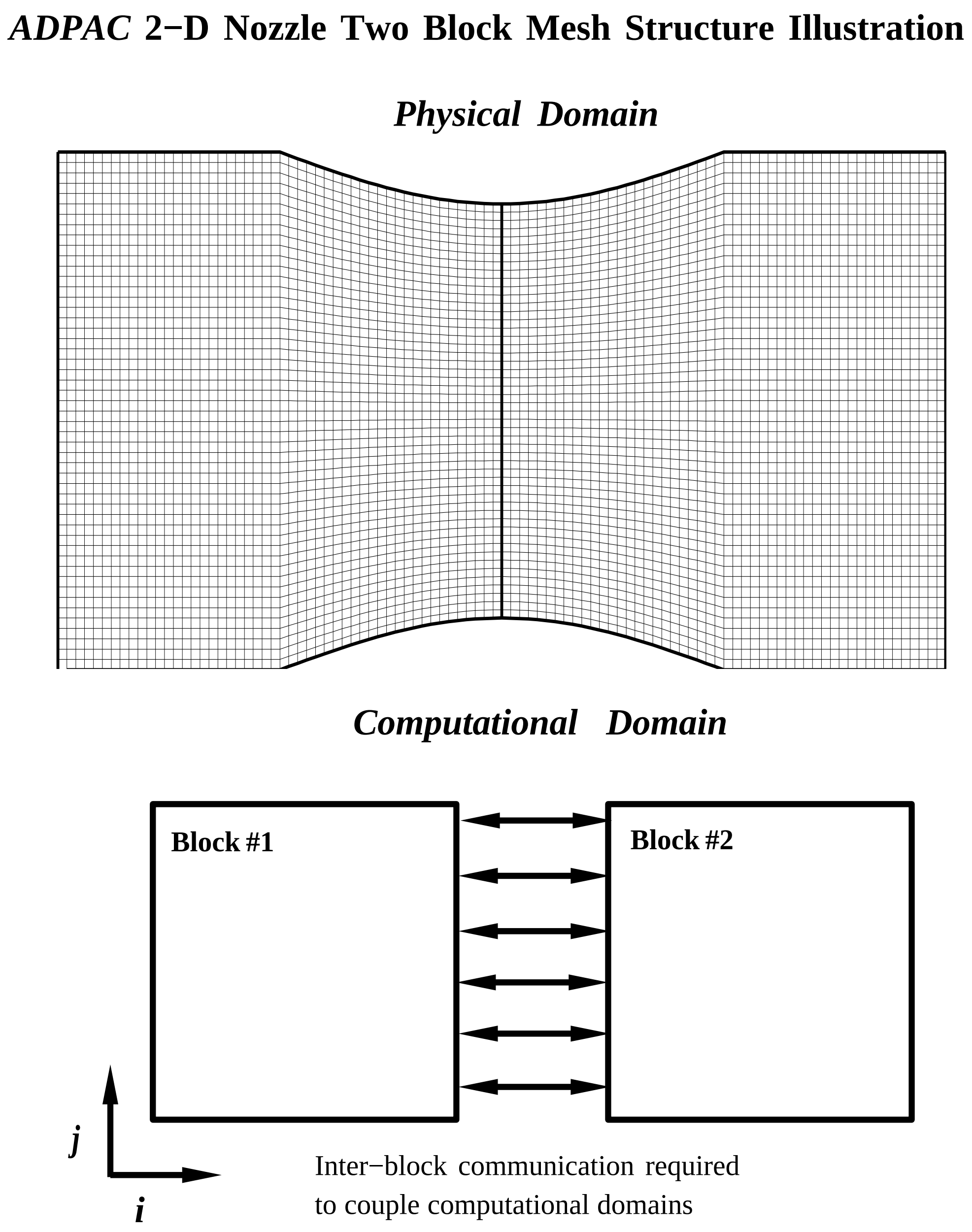

ADPAC 2−D Nozzle Two Block Mesh Structure Illustration
Physical Domain
Computational Domain
Block #1
Block #2
j
i
Inter−block communication required
to couple computational domains

Figure 2.2: *ADPAC08* 2-D Two Block Mesh Structure Illustration

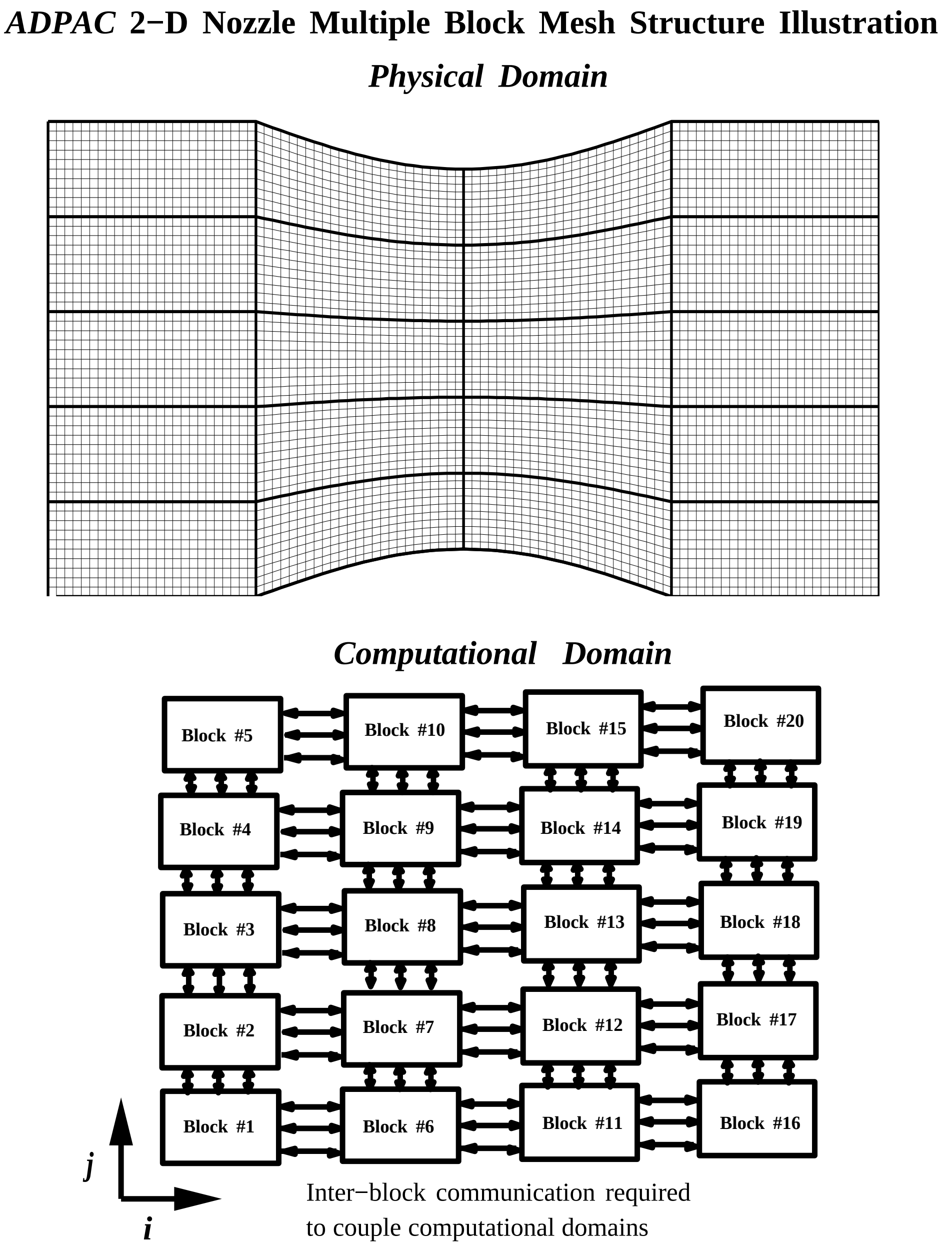


Figure 2.3: *ADPAC08* 2-D Multiple Block Mesh Structure Illustration

aerodynamic solution grows larger, the number of inter-domain communication paths increases in a corresponding manner. (It should be noted that this domain decomposition/communication overhead relationship is also a key concept in parallel processing for large scale computations, and thus, the *ADPAC08* code possesses a natural domain decomposition division for parallel processing afforded by the multiple-block grid data structure.

For the simple nozzle case illustrated in Figure 2.1 it would seem that there is no real advantage in using a multiple-block grid, and this is probably true. For more complicated geometries, such as the turbine vane coupled O-H grid system shown in Figure 2.4 and the corresponding computational domain communication scheme shown in Figure 2.5, it may not be possible to generate a single structured grid to encompass the domain of interest without sacrificing grid quality, and therefore, a multiple-block grid system has significant advantages.

The *ADPAC08* code utilizes the multiple-block grid concept to the full extent by permitting an arbitrary number of structured grid blocks with user specifiable communication paths between blocks. The inter-block communication paths are implemented as a series of boundary conditions on each block which, in some cases, communicate flow information from one block to another. The advantages of the multiple-block solution concept are exploited throughout the remainder of this document as a means of treating complicated geometries, multiple blade row turbomachines of varying blade number, and to exploit computational enhancements such as multigrid.

2.2 Multiple Blade Row Solution Concepts

Armed with an understanding of the multiple-block mesh solution concept discussed in the previous section, it is now possible to describe how this numerical solution technique can be applied to predict complicated flows. Specifically, this section deals with the prediction of flows through rotating machinery with multiple blade rows. Historically, the prediction of three-dimensional flows through multistage turbomachinery has been based on one of three solution schemes. These schemes are briefly illustrated and described in Figure 2.6.

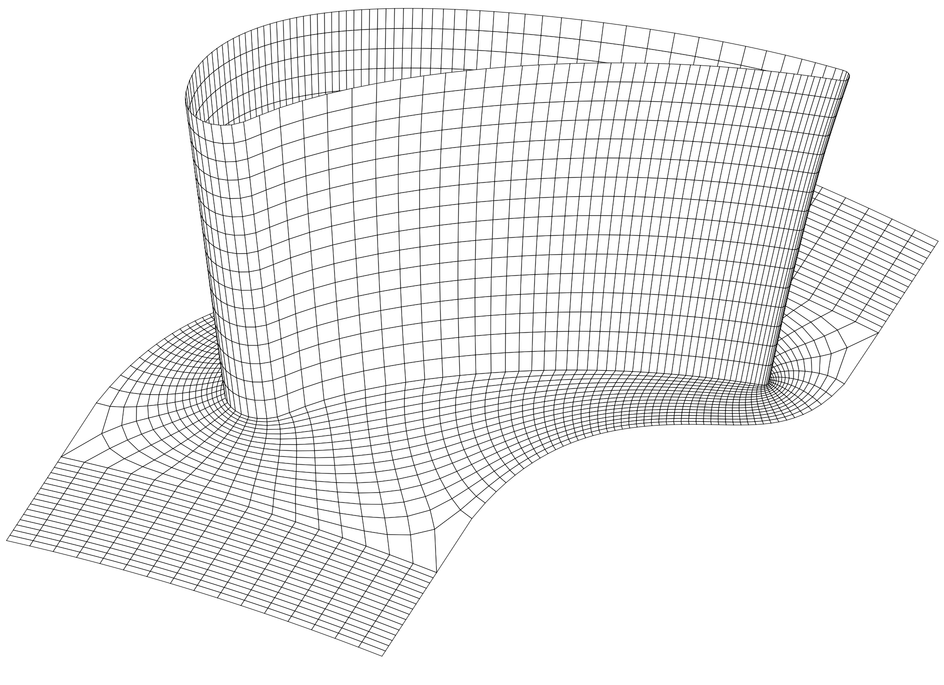


Figure 2.4: Coupled O-H Grid System for a Turbine Vane Cascade

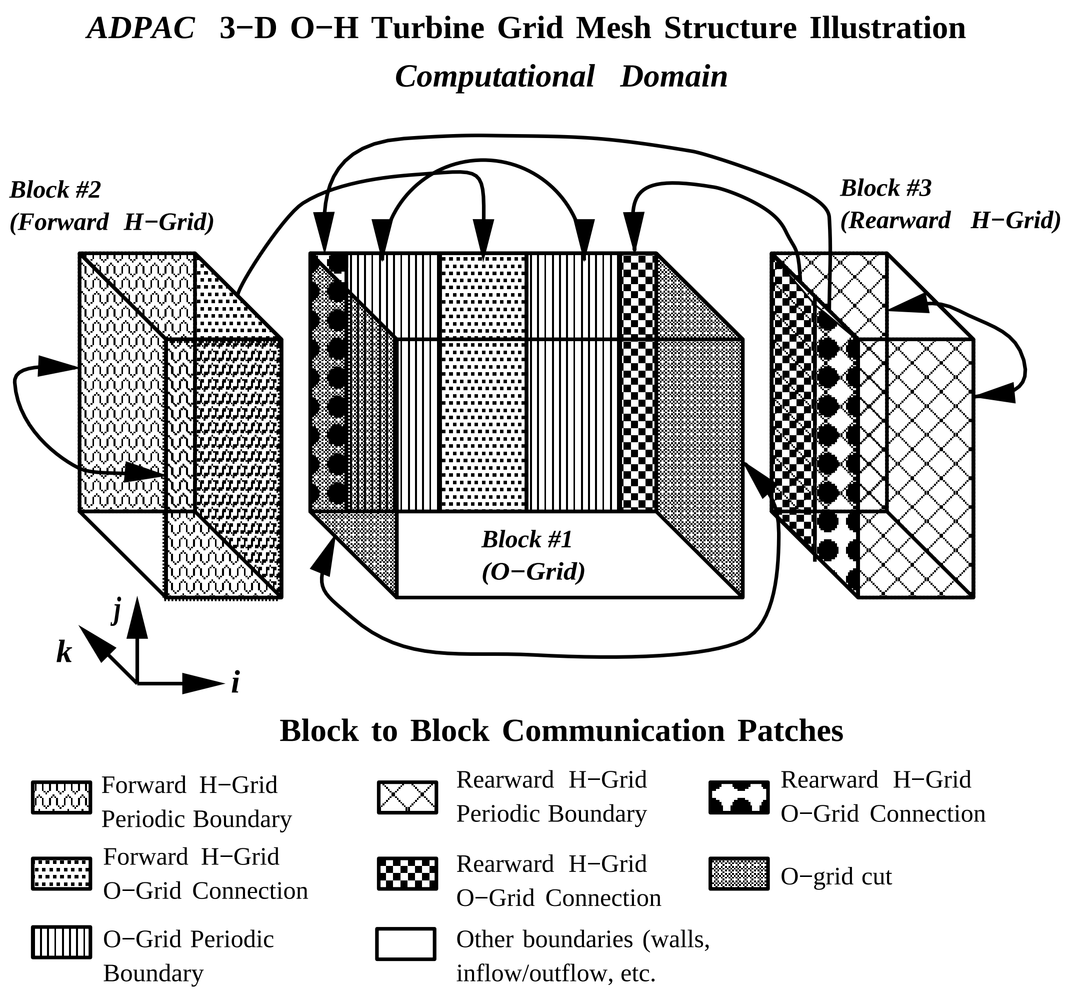

ADPAC 3−D O−H Turbine Grid Mesh Structure Illustration
Computational Domain
Block #2
(Forward H−Grid)
Block #3
(Rearward H−Grid)
Block #1
(O−Grid)
j
k
i
Block to Block Communication Patches
Forward H−Grid Periodic Boundary
Forward H−Grid O−Grid Connection
O−Grid Periodic Boundary
Rearward H−Grid Periodic Boundary
Rearward H−Grid O−Grid Connection
Other boundaries (walls, inflow/outflow, etc.
Rearward H−Grid O−Grid Connection
O−grid cut

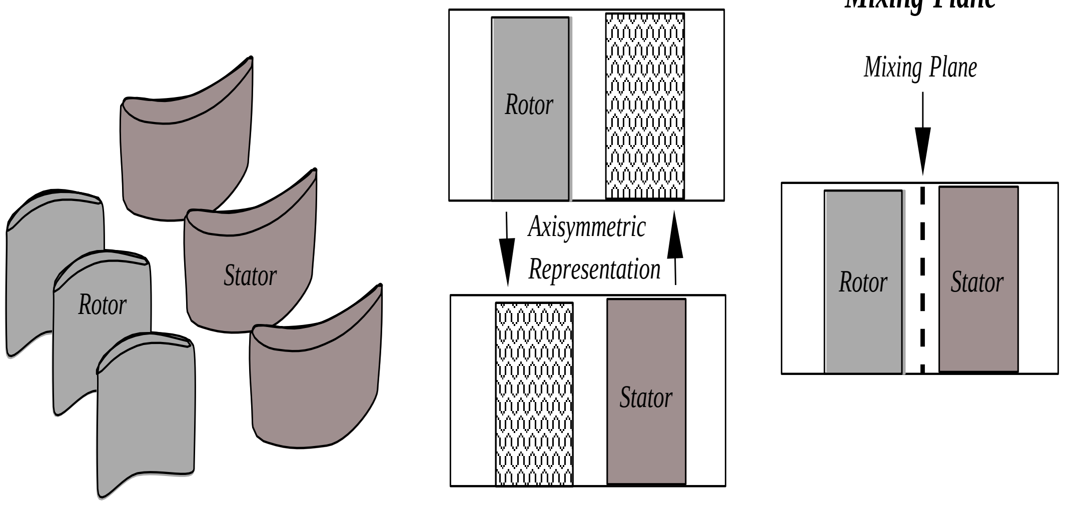
Figure 2.5: Computational Domain Communication Scheme for Turbine Vane O-H Grid System

Multiple Blade Row Numerical Solution Concepts

3–D Rotor/Stator Interaction

Average–Passage Simulation

Circumferential Mixing Plane



3–D time–dependent Navier–Stokes equations

Multiple blade passages for each blade row or phase–lagged boundaries

Time–dependent coupling of individual blade passage domains

Computationally expensive
Multiple blade passages per blade row

Average–passage equation system

3–D steady solution of entire domain for each blade row

Adjacent blade rows represented by blockage/body forces in 3–D solution

Solutions have common axisymmetric flowfield

Correlation model for mixing terms

Computational cost still rather high

Steady Navier Stokes solution

Computational domain limited to near blade region

Circumferential mixing plane provides inter–blade row communication

Lower computational cost

Figure 2.6: Multiple Blade Row Numerical Solution Schemes

The first scheme involves predicting the time-resolved unsteady aerodynamics resulting from the interactions occurring between relatively rotating blade rows. Examples of this type of calculation are given by Rao and Delaney [4], Jorgensen and Chima [5], and Rai [9]. This approach requires either the simulation of multiple blade passages per blade row, or the incorporation of a phase-lagged boundary condition to account for the differences in spatial periodicity for blade rows with dissimilar blade counts. Calculations of this type are typically computationally expensive, and are presently impractical for machines with more than 2-3 blade rows.

The second solution technique is based on the average-passage equation system developed by Adamczyk [6]. In this approach, separate 3-D solution domains are defined for each blade row which encompasses the overall domain for the entire turbomachine. The individual solution domains are specific to a particular blade row, although all blade row domains share a common axisymmetric flow. In the solution for the flow through a specific blade passage, adjacent blade rows are represented by their time and space-averaged blockage, body force, and energy source contributions to the overall flow. A correlation model is used to represent the time and space-averaged flow fluctuations representing the interactions between blade rows. The advantage of the average-passage approach is that the temporally and spatially averaged equations system reduce the solution to a steady flow environment; and, within the accuracy of the correlation model, the solution is representative of the average aerodynamic condition experienced by a given blade row under the influence of all other blade rows in the machine. The disadvantage of the average-passage approach is that the solution complexity and cost grow rapidly as the number of blade passages increases, and the validity of the correlation model is as yet unverified.

The third approach for the prediction of flow through multistage turbomachinery is based on the mixing plane concept. A mixing plane is an arbitrarily imposed boundary inserted between adjacent blade rows across which the flow is "mixed out" circumferentially. This circumferential mixing approximates the time-averaged condition at the mixing plane and allows the aerodynamic solution for each blade passage to be performed in a steady flow environment. The mixing plane concept was recently applied to realistic turbofan engine configurations by Dawes [7]. Flow variables on either side of the mixing plane are circumferentially averaged and passed to the neighboring blade row as a means of smearing out the circumferential nonuniformities resulting from dissimilar blade counts. The mixing plane concept is a much more cost-effective approach computationally because the flow is steady, and the individual blade passage domains are limited to a near-blade region. Unfortunately, the accuracy of this approach is clearly questionable under some circumstances because of the placement of the mixing plane and the loss of spatial information resulting from the circumferential averaging operator.

The *ADPAC08* program possesses features which permit multiple blade row solutions using either the time-dependent interaction approach or the mixing plane concept, described above. Average-passage simulations for realistic turbofan engine configurations were recently reported under Task IV of this contract, and further details on this approach can be found in Reference [8]. *ADPAC08* predictions utilizing the time-accurate rotor/stator interaction technique requires that a sufficient number of blade passages be represented in each row such that the circumferential distance represented in each blade row is constant. This limits the blade counts which can be effectively simulated through this technique. For example, for the simple single-stage calculation suggested in Figure 2.6, if the rotor has 36 blades and the stator has 48 blades, a time dependent solution would require, as a minimum, 3 rotor blade passages and 4 stator blade passages to accommodate the common circumferential pitch requirement. If the rotor has 35 blades, and the stator has 47 blades, however, then both blade rows would require that every blade passage be modeled as no simpler reduction in blade count is possible. This restriction will appear quite often, as turbomachinery designers often do not like to design neighboring blade rows with blade counts which have a common integer factor. Ultimately, this type of problem will require the incorporation of a phase-lagged boundary condition which would permit time-dependent interaction solutions for neighboring blades using only one blade passage per blade row.

If, instead, a mixing plane type of calculation is desired, then the multiple block scheme may again be invoked by utilizing a single blade passage per blade row, where each grid block has a common mating surface with a neighboring blade row. The only special requirement here is that boundary condition routines be available to adequately perform the circumferential averaging between blade rows and supply the block-to-block communication of this information in the multiple-block mesh solution algorithm. Section 3.7 describes the techniques for applying this type of boundary condition.

2.3 2-D/3-D Solution Zooming Concepts

A third unique feature of the *ADPAC08* solution system involves the concept of coupling two-dimensional and three-dimensional solution domains to obtain representative simulations of realistic high bypass ducted fan engine concepts. A complicating factor in the analysis of flows through turbofan engine systems results from the interactions between adjacent blade rows, and, in the case of a ducted fan, the effects of downstream blade rows on the aerodynamics of the upstream fan rotor. Historically, in the design of multistage turbomachinery, an axisymmetric representation of the flow through a given blade row has been used to effectively reduce the complexity of the overall problems to a manageable level. Similarly, an efficient approach to the numerical simulation of downstream blade rows could naturally utilize an axisymmetric representation of the effects of these rows through a two-dimensional grid system, with blade blockage, body force, and energy terms representing the axisymmetric averaged aerodynamic influence imparted by the embedded blade row. This concept is illustrated graphically in Figure 2.7 for a representative turbine stage.

A numerical solution of the flow through the fan rotor is complicated by the presence of the core stator, bypass stator, and bypass splitter. It is undesirable to restrict the solution domain to the fan rotor alone as this approach neglects the potential interactions between the fan rotor and the downstream geometry. The *ADPAC08* program permits coupled solutions of 3-D and 2-D mesh blocks with embedded blade row blockage, body force, and energy terms as a means of efficiently treating these more complicated configurations. Blade force terms may be determined from a separate 3-D solution, or may be directly specified based on simpler design system analyses. Neighboring 2-D and 3-D mesh blocks are numerically coupled through a circumferential averaging procedure which attempts to globally satisfy the conservation of mass, momentum and energy across the solution domain interface. The "dimensional zooming" capability permitted by the 2-D/3-D mesh coupling scheme is considered a vital asset for the accurate prediction of the flow through modern high-speed turbofan engine systems.

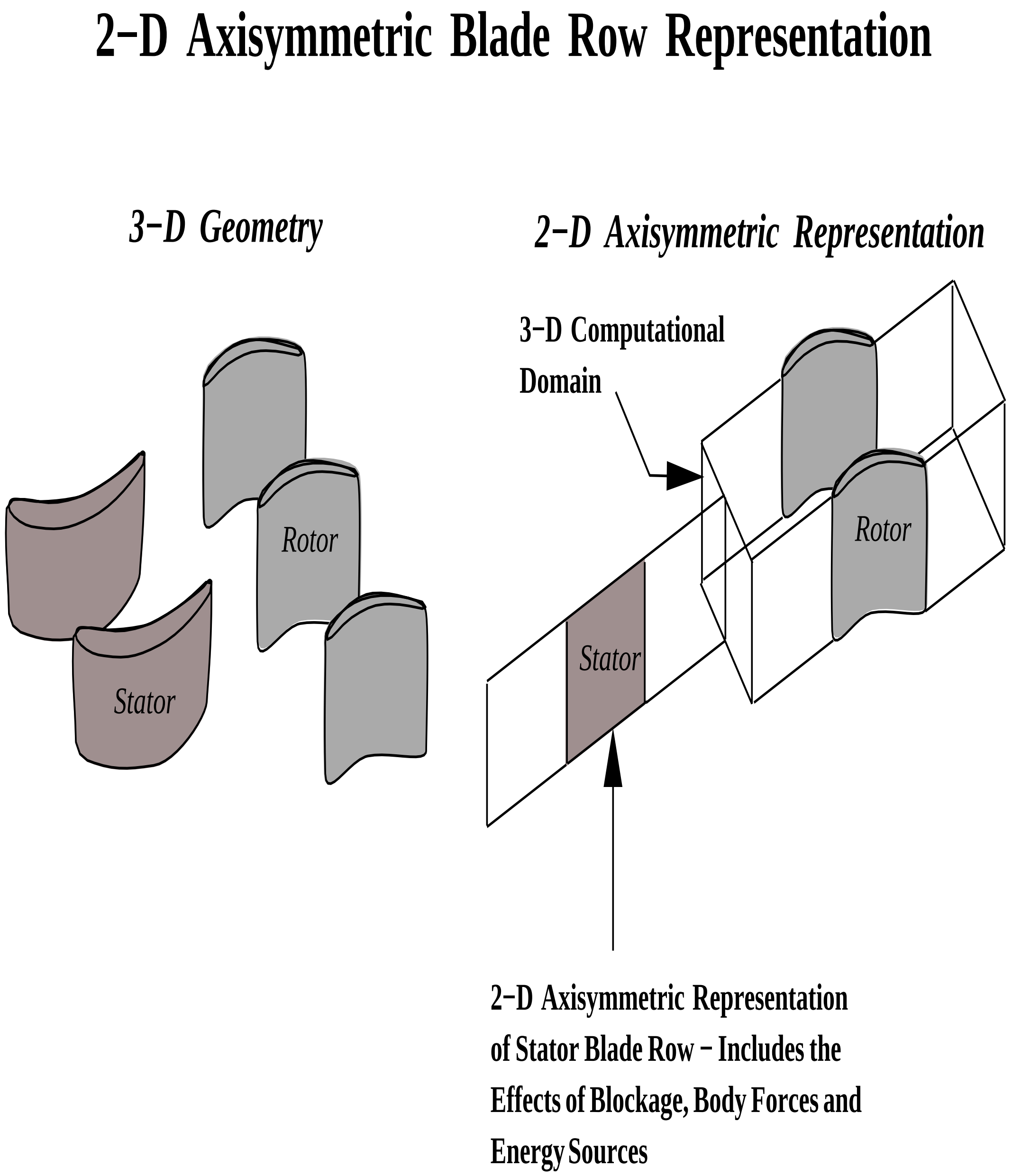
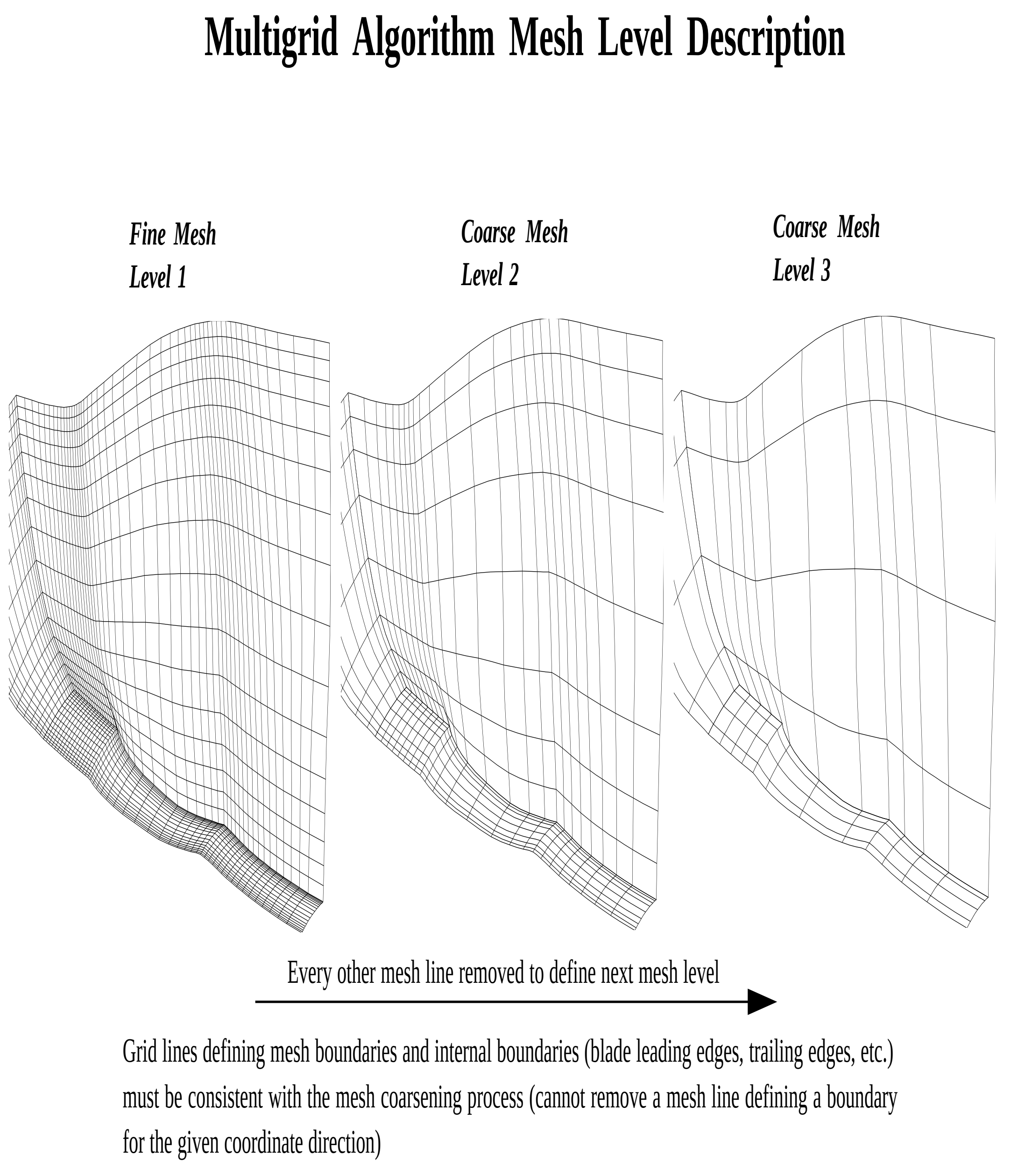

2−D Axisymmetric Blade Row Representation
3−D Geometry
2−D Axisymmetric Representation
3−D Computational Domain
Rotor
Stator
Rotor
Stator
2−D Axisymmetric Representation of Stator Blade Row − Includes the Effects of Blockage, Body Forces and Energy Sources

Figure 2.7: 2-D Axisymmetric Flow Representation of a Turbomachinery Blade Row

2.4 Multigrid Convergence Acceleration Concepts

For completeness, a brief section is included here to discuss the multigrid convergence acceleration solution technique incorporated into the *ADPAC08* code. Multigrid (please do not confuse this with a multiple-block grid!) is a numerical solution technique which attempts to accelerate the convergence of an iterative process (such as a steady flow prediction using a time-marching scheme) by computing corrections to the solution on coarser meshes and propagating these changes to the fine mesh through interpolation. This operation may be recursively applied to several coarsenings of the original mesh to effectively enhance the overall convergence. Coarse meshes are derived from the preceding finer mesh by eliminating every other mesh line in each coordinate direction as shown in Figure 2.8. As a result, the number of multigrid levels (coarse mesh divisions) is controlled by the mesh size, and, in the case of the *ADPAC08* code, by the mesh indices of the boundary patches used to define the boundary conditions on a given mesh block (see Figure 2.8). These restrictions suggest that mesh blocks should be constructed such that the internal boundaries and overall size coincide with numbers which are compatible with the multigrid solution procedure (i.e., the mesh size should be 1 greater than any number which can be divided by 2 several times and remain whole numbers; e.g. 9, 17, 33, 65 etc.) Further details on the application of the *ADPAC08* multigrid scheme are given in Section 3.6 and in Reference [1].

A second multigrid concept which should be discussed is the so-called "full" multigrid startup procedure. The "full" multigrid method is used to start up a solution by initiating the calculation on a coarse mesh, performing several time-marching iterations on that mesh (which, by the way could be multigrid iterations if successively coarser meshes are available), and then interpolating the solution at that point to the next finer mesh, and repeating the entire process until the finest mesh level is reached. The intent here is to generate a reasonably approximate solution on the coarser meshes before undergoing the expense of the fine mesh multigrid cycles. Again, the "full" multigrid technique only applies to starting up a solution, and therefore, it is not normally advisable to utilize this scheme when the solution is restarted from a previous solution as the information provided by the restart data will



Every other mesh line removed to define next mesh level

Grid lines defining mesh boundaries and internal boundaries (blade leading edges, trailing edges, etc.) must be consistent with the mesh coarsening process (cannot remove a mesh line defining a boundary for the given coordinate direction)

Figure 2.8: Multigrid Mesh Coarsening Strategy and Mesh Index Relation

likely be lost in the coarse mesh initialization.

2.5 General Solution Procedure Sequence

The *ADPAC08* code is distributed as a compressed *tar* file which must be processed before the code may be utilized. The instructions in Appendix A describe how to obtain the distribution file, and extract the necessary data to run the code. This operation is typically required only once when the initial distribution is received. Once the source files have been extracted, the sequence of tasks listed below are typical of the events required to perform a successful analysis using the *ADPAC08* code.

Step 1.) Define the problem:

This step normally involves selecting the geometry and flow conditions, and defining which specific results are desired from the analysis.The definition of the problem must involve specifying whether steady state or time-dependent data are required, whether an inviscid calculation is sufficient, or whether a viscous flow solution is required, and some idea of the relative merits of solution accuracy versus solution cost (CPU time) must be considered.

Step 2.) Define the geometry and flow domain:

Typically, geometric features such as airfoils, ducts, and flowpath endwalls are required to geometrically define a given problem. The solution domain may be chosen to include the external flow, internal engine passage flows, and/or leakage flows. The flow domain is normally defined large enough such that the region of interest is far enough away from the external boundaries of the problem to ensure that the solution is not unduly influenced by the external boundary conditions.

Step 3.) Define a block structure:

Once the geometry and solution domain has been numerically defined, the implementation of the solution mesh structure must be considered. This process begins with a determination of the domain block structure, if and when more than one mesh block is required for a given solution. The possibility of incorporating 2-D mesh blocks should be considered whenever possible due to the computational savings afforded by this approach (see Section 2.3).

For convenience, several standard block structures for turbomachinery geometries have been constructed and will be discussed later in this document (see Chapter 5). This should not discourage users from developing their own block structures should specific needs be unresolved by the standard block structures.

Step 4.) Generate a numerical grid for the domain of interest:

Most of the standard grid block structures defined in this document can be adequately handled through either the *TIGG3D* [15] or the *CHGRIDV2* [2] grid generation programs. Other grid generation programs may be equally useful, and a conversion program called *MAKEADGRID* (described in Chapter 7) is included to convert non-standard meshes into *ADPAC08* format.

Step 5.) Generate a standard input file:

The standard input file controls operations specific to a particular run of the *ADPAC08* code. Options such as the number of iterations, damping parameters, and input/output control of the code execution may all be governed by the values specified in the standard input file. For the standard block structures described above, a program called *SETUP* (see Chapter 8) is provided which will interactively query the user about the desired run for one of the standard block configurations, and construct an appropriate input file.

Step 6.) Generate a boundary data file:

The boundary data file controls the application of boundary conditions on the grid block structure provided to the flow code. The boundary data specifications are specific to the mesh being used in a given calculation. For the standard block structures described above, a program called *SETUP* (see Chapter 8) is provided which will interactively query the user about the desired run for one of the standard block configurations, and construct an appropriate boundary data file. For other block configurations, the user must construct the boundary data file by hand according to the format described in Section 3.7.

Step 7.) Run *ADPAC08* to predict the aerodynamics:

Chapter 3 is available to describe the commands necessary to perform this task. A sample test case is also completely outlined in Appendix A. In many cases, a given calculation will involve several applications of the *ADPAC08* code, restarted from the previous calculation as a means of breaking up a large problem into several shorter calculations.

Step 8.) Plot and process the results:

An interactive post processing program called *ADPOST* is provided to handle tasks such as mass-averaging flow variables to simplify the interpretation of the computed results (see Chapter 4). Output data is also provided for widely available plotting programs such as *PLOT3D* [11] and *FAST* [13].

A condensed description of the commands involved in the steps described above beginning with extracting the source code from the distribution, compiling the codes, setting up a case, and running a case, is given in the Appendix. Separate sections are provided in the chapters which follow to describe in detail the basis and operation of the codes used in the steps above.

It is worthwhile mentioning that the development and application of the codes described in this manual were performed on Unix-based computers. All files are stored in machine-independent format. Small files utilize standard ASCII format, while larger files, which benefit from some type of binary storage format, are based on the Scientific DataBase Library (SDBLIB) format [10]. The SDBLIB format utilizes machine-dependent input/output routines which permit machine independence of the binary data file. The SDBLIB routines are under development at the NASA Lewis Research Center.

Most of the plotting and graphical postprocessing of the solutions was performed on graphics workstations. The *PLOT3D* [11], and *FAST* [13] graphics software packages developed at NASA Ames Research Center were extensively used for this purpose, and data files for these plotting packages are generated automatically. These data files are written in what is known as *PLOT3D* multiple-grid format. (See *ADPAC08* File Description, Section 3.5).

2.6 Consolidated Serial/Parallel Code Capability

One of the practical difficulties of performing CFD analyses is finding sufficient computational resources to allow for adequate modeling of complex geometries. Oftentimes, workstations are not large enough, and supercomputers have either long queues, high costs, or both. Clearly, a means of circumventing these difficulties without giving up the flexibility of the CFD code or the complexity of the model would be welcome. One possibility is to write a code which could run in parallel across a number of processors, with each one having only a piece of the problem. Then, a number of lesser machines could be harnessed together to make a virtual supercomputer.

The most likely candidates for creating such a machine are the workstations which are fully loaded during the day, but sit idle at night. Tremendous power could be made available at no extra cost. There are also massively parallel computers available on the market designed specifically for such applications. These machines are aiming at order of magnitude improvements over present supercomputers.

The problem of course, lies in the software. Parallelization is today about as painful as vectorization was a decade ago. There is no standard parallel syntax, and no compiler to automatically parallelize a code. It is difficult to write a parallel code which is platform independent. What makes things worse is that there is no clear leader in the parallel computer industry, as there has been in the supercomputer industry.

The objective behind the development of the consolidated *ADPAC08* code described in this manual was to create a platform independent parallel code. The intent was to design a parallel code which looks and feels like a traditional code, capable of running on networks of workstations, on massively parallel computers, or on the traditional supercomputer. User effort was to be minimized by creating simple procedures to migrate a serial problem into the parallel environment and back again.

2.7 Parallelization Strategy

The *ADPAC08* code has some innate advantages for parallelization: it is an explicit, multi-block solver with a very flexible implementation of the boundary conditions. This presents two viable options for parallelization: parallelize the internal solver (the "fine-grained" approach), or parallelize only the boundary conditions (the "coarse-grained" approach). The fine-grained approach has the advantage that block size is not limited by processor size. This is the approach frequently taken when writing code for massively parallel computers, which are typically made up of many small processors. The coarse-grained approach is favored when writing code for clusters of workstations, or other machines with a few large processors. The dilemma is that a parallel *ADPAC08* needs to run well on both kinds of machines.

The fine grained approach is especially enticing for explicit solvers. Explicit codes have proven to be the easiest to parallelize because there is little data dependency between points. For a single block explicit solver, fine-grained parallelization is the clear choice. However, with a multiblock solver, the boundary conditions must be parallelized in addition to the interior point solver, and that can add a lot of programming effort. The coarse-grained approach is admittedly easier for multi-block solvers, but what if the blocks are too big for the processors? The simplest answer is to require the user to block out the problem so that it fits on the chosen machine. This satisfies the programmer, but the user is faced with a tedious chore. If the user decides to run on a different machine, then the job may have to be redone. The pain saved by the programmer is passed directly to the user.

A compromise position was reached for parallel *ADPAC08* code. The coarse-grained approach is used, but supplemental tools are provided to automatically generate new grid blocks and boundary conditions for a user-specified topography. In this way, the parallel portions of the code are isolated to a few routines within *ADPAC08* , and the user is not unduly burdened with architecture considerations. Details of running the *ADPAC08* in parallel are given in a later chapter.

Chapter 3

ADPAC08 : 3-D EULER/NAVIER-STOKES FLOW SOLVER OPERATING INSTRUCTIONS

3.1 Introduction to *ADPAC08*

This chapter contains the operating instructions for the *ADPAC08* time-dependent multiple grid block 3-D Euler/Navier-Stokes aerodynamic analysis. These instructions include some general information covering executing the code, defining array limits, compiling the flow solver, setting up input files, running the code, and examining output data. The *ADPAC08* flow solver source programs are written in FORTRAN 77, and have been used successfully on Cray UNICOS and IBM VM/CMS mainframe computer systems as well as IBM AIX Operating System and Silicon Graphics 4D workstations using a UNIX operating system.

3.2 General Information Concerning the Operation of the *ADPAC08* Code

Approximate computational storage and CPU requirements for the *ADPAC08* code can be conservatively estimated from the following formulas:

$$\begin{aligned} \text{CPU sec} &\approx 1.0x10^{-4}(\#\text{ grid points})(\#\text{ iterations}) \\ \text{Memory MW} &\approx 6.8x10^{-5}(\#\text{ grid points}) \end{aligned}$$

These formulas are valid for a Cray-YMP computer operating under the UNICOS environment and the cf77 compiler, version 6.0.11 and above. The times reported are for a single processor only, and are not indicative of any parallelization available through the Cray autotasking or microtasking facilities. Without multigrid, steady inviscid flow calculations normally require approximately 2000 iterations to reduce the maximum residual by three orders of magnitude (10^3) which is normally an acceptable level of convergence for most calculations. Viscous flow calculations generally require 3000 or more iterations to converge. When multigrid is used, the number of iterations required to obtain a converged solution is often one third to one fourth the number of iterations required for a non-multigrid calculation. Convergence for a viscous flow case is generally less well behaved than a corresponding inviscid flow calculation, and in many cases, it is not possible to reduce the maximum residual by three orders of magnitude due to oscillations resulting from vortex shedding, shear layers, etc. A determination of convergence for a viscous flow case must often be based on observing the mass flow rate or other global parameter, and terminating the calculation when these variables no longer change. The number of iterations required for an unsteady flow calculation is highly case-dependent, and may be based on mesh spacing, overall time-period, complexity of the flow, etc.

The *ADPAC08* program produces output files suitable for plotting using the *PLOT3D* [11], *SURF* [12], and *FAST* [13] graphics software packages developed at the NASA Ames Research Center. *PLOT3D* format data files are written for both absolute and relative flows (see Chapter 2 for a description of the *PLOT3D* format). The user may also elect to have additional *PLOT3D* absolute flow data files output at constant iteration intervals during the course of the solution. These files may be used as instantaneous flow "snapshots" of an unsteady flow prediction.

3.3 Configuring *ADPAC08* Maximum Array Dimensions

The first step required before attempting to compile and run the *ADPAC08* program is to set the maximum array size required for the analysis prior to the compilation process. The maximum array size will ultimately determine the largest problem (in terms of total number of mesh points) which can be run with the code. The larger the array limits, the larger the number of grid points which may be used. Unfortunately, setting larger array limits also increases the total amount of memory required by the program, and hence, can impede the execution of the code on memory-limited computing systems. Ideally, the code should be dimensioned just large enough to fit the problem at hand. It should be mentioned that storage requirements are dependent on whether the multigrid convergence acceleration technique is used or not. This dependency is explained in more detail in the paragraphs which follow. Approximate total computational storage and CPU requirements can be estimated for the *ADPAC08* aerodynamic analysis from the formulas listed in Section 3.2.

Array dimensions are specified in the *ADPAC08* program by a set of FORTRAN PARAMETER statements. The array limits are specified in the source code file **parameter.inc**. A sample **parameter.inc** file is given below:

```
PARAMETER( NBMAX  = 10 )
PARAMETER( NRA3D  = 250000  )
PARAMETER( NBL2D  = 20000   )
PARAMETER( NRA1D  = 1000    )
PARAMETER( NBCPBL = 15      )
PARAMETER( NRAINT = 4000    )
PARAMETER( NBFRA  = 12000   )
PARAMETER( LGRAFX = 4000    )
PARAMETER( NSYST  = 1       )
PARAMETER( NBFFILE= 10      )
PARAMETER( NBCNT1 = 1000    )
```

Each statement in the **parameter.inc** file is ultimately embedded in every subroutine through a FORTRAN **include** statement.

During execution, the *ADPAC08* program automatically checks to make sure enough storage is available for all the blocks and issues a fatal error message if an array size is exceeded.

Before proceeding with a description of the various parameter variables, it should be mentioned that a computational tool is available called *ADSTAT* which will read in an *ADPAC08* ready mesh file and determine the required parameter sizes for either a multigrid or non-multigrid run. The *ADSTAT* program is described in more detail in Chapter 10.

The various PARAMETER variables utilized in the **parameter.inc** file are described below.

NBMAX
The parameter **NBMAX** defines the maximum number of grid blocks permitted during execution of the *ADPAC08* multiple block solver. This number must be large enough to include every level of coarse mesh blocks created during a multigrid run. The *ADPAC08* code exploits the multiple block mesh structure during multigrid runs by creating and storing coarse mesh blocks from the corresponding fine mesh blocks. In other words, if it is intended to run a 5 block mesh with 3 levels of multigrid, then the parameter **NBMAX** must be at least 15.

NRA3D
The parameter **NRA3D** defines the maximum number of computational cells permitted for the finite volume time-marching algorithm. This parameter defines the maximum total number of mesh points (including multigrid coarse meshes, when applicable) which are permitted during an *ADPAC08* run. The minimum value for the **NRA3D** parameter for a given mesh system may be calculated as follows:

$$NRA3D \geq \sum_{m=1}^{m=NBLKS} [(IMX)_m + 1][(JMX)_m + 1][(KMX)_m + 1]$$

where $(IMX)_m$, $(JMX)_m$, and $(KMX)_m$ indicate the number of mesh points in the i, j, and k mesh coordinate directions, respectively, for mesh block m, and $NBLKS$ is the total number of grid blocks. Sample calculations of the minimum value for the **NRA3D** parameter for a multiple block mesh are provided below.

Suppose we intend to perform a solution on a mesh consisting of 3 mesh blocks with $49x17x17$, $25x17x17$, and $129x33x49$ mesh points, respectively. For a non-multigrid calculation, the total number of mesh blocks is simply 3, and the minimum value for parameter **NRA3D** may be computed as:

$$NRA3D = (49+1)(17+1)(17+1) + (25+1)(17+1)(17+1)$$

$$+(129+1)(33+1)(49+1) = 245,624$$

If, using the same mesh system, it is desired to employ 3 levels of multigrid, additional storage must also be allocated for the coarse mesh systems, and the minimum value for parameter **NRA3D** must be recomputed as:

$$(NRA3D)_1 = (49+1)(17+1)(17+1) + (25+1)(17+1)(17+1)$$

$$+(129+1)(33+1)(49+1) = 245,624$$

$$(NRA3D)_2 = (25+1)(9+1)(9+1) + (13+1)(9+1)(9+1)$$

$$+(65+1)(17+1)(25+1) = 34,888$$

$$(NRA3D)_3 = (13+1)(5+1)(5+1) + (7+1)(5+1)(5+1)$$

$$+(33+1)(9+1)(13+1) = 5,552$$

$$NRA3D = (NRA3D)_1 + (NRA3D)_2 + (NRA3D)_3 = 286,064$$

The requirement that the parameter variable **NRA3D** (and others) be based on array sizes 1 element larger than the grid dimensions results from the use of phantom points outside the computational domain to impose the numerical boundary conditions.

NBL2D

The parameter **NBL2D** is used to define the size of the temporary 2-D arrays utilized during the advancement of the time-marching algorithm for a given mesh block. As such, the parameter is based on the largest single dimension of *any* mesh block (2-D or 3-D) and may be determined by the following formula:

$$NBL2D \geq (max_{m=1,NBLKS}[(IMX)_m + 1, (JMX)_m + 1, (KMX)_m + 1])^2$$

where the variables $IMX, JMX, KMX, NBLKS$ are defined in the section describing **NRA3D** above.

Returning to the example mesh system utilized in the description of the parameter **NRA3D**, the minimum value for the parameter **NBL2D** may be computed as:

$$NBL2D = (129 + 1)^2 = 16900$$

This value is unchanged regardless of the number of multigrid levels since coarser meshes always result in smaller mesh sizes.

NRA1D

The parameter **NRA1D** is used to define the size of several 1-D arrays used to do various bookkeeping operations during the execution of the *ADPAC08* code. As such, the parameter is based on the sum of the maximum single dimension of all mesh blocks in the following manner:

$$NRA1D \geq \sum_{m=1}^{m=NBLKS} max[(IMX)_m + 1, (JMX)_m + 1, (KMX)_m + 1]$$

Returning to the example mesh system utilized in the description of the parameter **NRA3D**, the minimum value for the parameter **NRA1D** for a non-multigrid run be determined as:

$$NRA1D = (49 + 1) + (25 + 1) + (129 + 1) = 206$$

and for a 3 level multigrid run as:

$$(NRA1D)_1 = (49 + 1) + (25 + 1) + (129 + 1) = 206$$

$$(NRA1D)_2 = (25 + 1) + (13 + 1) + (65 + 1) = 106$$

$$(NRA1D)_3 = (13 + 1) + (7 + 1) + (33 + 1) = 56$$

$$NRA1D = (NRA1D)_1 + (NRA1D)_2 + (NRA1D)_3 = 368$$

NBCPBL

The parameter **NBCPBL** is used to define the size of the arrays used to store the boundary condition specifications for a given *ADPAC08* run. Since the number of boundary conditions normally scales according to the number of mesh blocks (as a minimum, 6 boundary conditions are required for each mesh block, see Section 3.7), the parameter **NBCPBL** implies the maximum number of boundary conditions per block, and the overall number of boundary conditions is determined by multiplying the parameters **NBMAX** and **NBCPBL**. It should be noted that a single block can, in fact, possess more than **NBCPBL** boundary condition specifications as long as the *total* number of boundary condition specifications for the entire problem does not exceed $NBMAX * NBCPBL$.

NRAINT
The parameter **NRAINT** is used to define the size of the temporary arrays used to store interpolation data for the non-contiguous mesh patching boundary condition specification **PINT**, described in Section 3.7. The **PINT** specification controls the numerical coupling between two mesh blocks possessing non-contiguous mesh boundaries which lie on a common surface. The numerical scheme utilizes a rather simple interpolation scheme based on an electrical circuit analogy, and stores the "nearest neighbors" for each mesh point to avoid the expense of constantly searching for the interpolation stencil between the two mesh surfaces. Determining the value required for the parameter **NRAINT** is normally performed by summing up all of the mesh elements involved in all of the **PINT** specifications (including coarse mesh specifications from a multigrid run). For example, if two meshes with non-contiguous mesh boundaries of 49x33 and 25x17 are being updated using the **PINT** specification, then the minimum value for the **NRAINT** parameter for a nonmultigrid run would be determined as:

$$NRAINT = (49-1)(33-1) + (25-1)(17-1) = 1920$$

In this case, the **NRAINT** parameter is based on the mesh indices minus one, since the storage in the finite volume solver is actually based on the number of mesh *cells*, not the number of mesh points, even though the boundary specification is based on actual *mesh* indices. The equivalent value for a run utilizing 3 levels of multigrid would be:

$$(NRAINT)_1 = (49-1)(33-1) + (25-1)(17-1) = 1920$$

$$(NRAINT)_2 = (25-1)(17-1) + (13-1)(9-1) = 480$$

$$(NRAINT)_3 = (13-1)(9-1) + (7-1)(5-1) = 120$$

$$NRAINT = (NRAINT)_1 + (NRAINT)_2 + (NRAINT)_3 = 2520$$

Naturally, if additional **PINT** specifications are employed then the contributions from these specifications must also be added to the total.

NBFRA

The parameter **NBFRA** is used to define the size of the 2-D arrays used to store the blade element blockage, body force, and energy source terms for the 2-D block solution scheme. Since these arrays are utilized for any 2-D mesh block regardless of whether blade element blockage and source terms are utilized, the arrays must be dimensioned large enough to store all the elements of all of the 2-D mesh blocks (including coarse meshes for multigrid runs) much in the manner that **NRA3D** is used to store all of the elements of all of the mesh blocks. Mathematically, the minimum value for the parameter **NBFRA** may be calculated as:

$$NBFRA \geq \sum_{m=1}^{m=NBLKS} [(IMX)_m + 1][(JMX)_m + 1] L_{2D}(m)$$

where the variables IMX, JMX, KMX and $NBLKS$ are described in the definition of parameter **NRA3D**, above. The variable $L_{2D}(m)$ is a trigger to indicate whether the grid block m is 2-D (1) or 3-D (0). For example, suppose a multiple block solution is being performed for a mesh system comprised of two 2-D meshes sized 49x25x1 and 33x17x1 and a 3-D mesh sized 33x25x29. For a non-multigrid run, the minimum value for the parameter **NBFRA** may be calculated as:

$$NBFRA = (49{+}1)(25{+}1)(1){+}(33{+}1)(17{+}1)(1){+}(129{+}1)(25{+}1)(0) = 1912$$

and for a run employing 3 levels of multigrid as:

$$(NBFRA)_1 = (49{+}1)(25{+}1)(1){+}(33{+}1)(17{+}1)(1){+}(129{+}1)(25{+}1)(0) = 1912$$

$$(NBFRA)_2 = (25{+}1)(13{+}1)(1){+}(17{+}1)(9{+}1)(1){+}(65{+}1)(13{+}1)(0) = 544$$

$$(NBFRA)_3 = (13+1)(7+1)(1) + (9+1)(5+1)(1) + (33+1)(7+1)(0) = 172$$

$$NBFRA = (NBFRA)_1 + (NBFRA)_2 + (NBFRA)_3 = 2628$$

LGRAFX

The parameter **LGRAFX** is used to define the size of the temporary 3-D arrays used for the run-time graphics display option available in the *ADPAC08* code. If the run-time graphics option is employed, then the parameter **LGRAFX** can be determined in the same manner as the parameter

NRA3D. If the run-time graphics option is not employed, then the parameter **LGRAFX** should be set to 1, resulting in a considerable savings in computational storage.

NSYST
The parameter **NSYST** is used to define the size of a character array which stores system call commands during the execution of the boundary condition routine **SYSTEM** (see Section 3.7). Normally, this is not used and may be set to a value of 1 to minimize storage. If the **SYSTEM** boundary routine is used, then **NSYST** must be at least as large as the number of **SYSTEM** boundary specifications in the *ADPAC08* boundary data file.

NBFFILE
The parameter **NBFFILE** is used to define the size of a character array which stores body force file names specified by the input variable **BFFILE** (see Section 3.6). Normally, this is not used and may be set to a value of 1 to minimize storage. If the **BFFILE** input variable is used, then **NBFFILE** must be at least as large as **NBMAX**.

NBCNT1
The parameter **NBFFILE** is used to define the size of the arrays used to save the interpolation stencils used in the **BCINT1** and **BCINTM** non-aligned mesh boundary coupling schemes. In an effort to computational and communication efficiency, the interpolation stencils used to update the non-aligned boundaries in these boundary condition routines are only calculated on the first step, and are subsequently saved to eliminate any redundant calculation. The **NBCNT1** parameter must be at least as large as the sum of the total number of points along all **BCINT1** and **BCINTM** non-aligned boundary patches. If **BCNT1** is set to 1, then the interpolation stencil saving feature is disabled, and the interpolation stencil is recalculated at every time step.

3.4 *ADPAC08* Compilation Using *Makefile*

Compilation of the *ADPAC08* source code into an executable form is handled through a UNIX-based *Makefile* facility. A *Makefile* is included with the standard distribution which permits automatic compilation of the code for several operational capabilities (both serial and parallel) and computer systems. The format of the *Makefile* compiling command is described below.

Several items should be mentioned prior to detailed discussion on the actual *Makefile* utilities. Section 3.5 describes the format of the binary files using the Scientific Database Library developed at NASA-Lewis [10]. The original version of the Scientific Database Library was found to be rather slow on some machines, and an equivalent limited capability C-based library was developed to accelerate the I/O processing in the code. This library is referred to as CSDB, and separate options for utilizing the CSDB library are included in the *Makefile*. In addition, the consolidated code is capable of both serial and parallel operation depending on the *Makefile* operation selected.

In the directory containing the FORTRAN source of the *ADPAC08* code, compilation is performed by executing the command:

make *option*

The **make** command is standard on UNIX systems and automatically interrogates the file *Makefile* for instructions on how to perform the compilation. The *option* argument may be any of the variables listed below:

No argument - same as *link* below.

help This option lists and describes all available *makefile* options. No executable is created.

link This is the standard UNIX system compilation for the serial version of the *ADPAC08* code. All non-standard programming constructs are avoided (such as graphics, or multi-processor features). This option will deliver a working executable on most UNIX systems which support standard naming conventions (*f77* as the standard compiler, etc.). The compilation includes basic compiler optimization (f77 -O). The executable name is simply **adpac**.

csdb This is the same as *link* above, except that the C-based scientific database libarary is linked instead of the standard scientific database libarary (which is thought to be slower). Prior to performing this compilation, the appropriate *make* command must be issued in the CSDB directory to assemble the CSDB library for the local machine. The executable name is simply **adpac**.

parallel This is the standard UNIX system compilation for the parallel version of the *ADPAC08* code. The standard APPL message passing library is incorporated, and therefore creation of this executable requires that a *make* has been issued in the APPL directory on the current machine. The *parallel* code may only be executed using the APPL *compute* function with a corresponding APPL *procdef* file. Prior to performing this compilation, the appropriate *make* command must be issued in the APPL directory to assemble the APPL library for the local machine. The executable name is simply **adpacp**.

parallel_csdb This is the same as *parallel* above, except that the C-based scientific database libarary is linked instead of the standard scientific database libarary (which is thought to be slower). Prior to performing this compilation, the appropriate *make* command must be issued in the CSDB and APPL directories to assemble the CSDB and APPL libraries for the local machine. The executable name is simply **adpacp**.

pfa This option is used on Silicon Graphics computers supporting the Power FORTRAN compiler option. Power FORTRAN is a Silicon Graphics product which does automatic multiprocessor compilation and is therefore not related to the *ADPAC08* message-passing parallelization strategy. The *pfa* compiled code is therefore still operated as a serial code, although may execute on multiple processors for Silicon Graphics workstations. The number of processors used is set by the NUM_THREADS environment variable. The compilation includes basic compiler optimization (f77 -O). The executable name is simply **adpac**.

csdb_pfa This option is the same as *pfa* above, except that the C-based scientific database libarary is linked instead of the standard scientific database libarary (which is thought to be slower). Prior to performing this compilation, the appropriate *make* command must be issued in the CSDB directory to assemble the CSDB library for the local machine. The executable name is simply **adpac**.

graphics This option compiles *ADPAC08* with the necessary routines needed to permit interactive graphics between network connected Silicon Graphics workstations. This option will only work when compiling on a Silicon Graphics workstation with IRIX operating system 4.0.1 or above. The full Silicon Graphics shared graphics libraries and X-windows system graphics libraries must be installed on the compiling workstation in order for this option to work. This feature is not recommended as it generally decreases performance and other visualization techniques are likely to produce more desirable results. The executable name is simply **adpac**.

csdb_graphics This option is the same as *graphics* above, except that the C-based scientific database libarary is linked instead of the standard scientific database libarary (which is thought to be slower). Prior to performing this compilation, the appropriate *make* command must be issued in the CSDB directory to assemble the CSDB library for the local machine. The executable name is simply **adpac**.

dbx This option is used for generating an executable version of the serial code which is compatible with the standard UNIX *dbx*-based debugging facility. This should work on any standard UNIX machine which supports *dbx* (Note: the code will run much more slowly when compiled in this fashion.) This option is used mainly for code development or debugging. The executable name is simply **adpac.dbx**.

cray This option is utilized when compiling the standard code on a Cray computer (implies a serial code). For best performance, the aggressive optimization option of the Cray compiler has been invoked (cf77 -Zv -Wf"-o aggress"). The executable name is simply **adpac.cray**.

cray_dbx This option is used for generating an executable version of the code which is compatible with the Cray *cdbx* debugging facility. (Note: the code will run much more slowly when compiled in this fashion.) This option is used mainly for code development or debugging. The executable name is simply **adpac.cray.dbx**.

aix This option is used when compiling the standard serial code on an IBM RS-6000 workstation running the AIX operating system. The executable name is simply **adpac.aix**.

aix_csdb This option is identical to *aix* above, except that the C-based scientific database libarary is linked instead of the standard scientific database libarary (which is thought to be slower). Prior to performing this compilation, the appropriate *make* command must be issued in the CSDB directory to assemble the CSDB library for the local machine. The executable name is simply **adpac.aix**.

aix_dbx This option is used for generating an executable version of the code which is compatible with the IBM AIX *dbx* debugging facility. (Note: the code will run much more slowly when compiled in this fashion.) This option is used mainly for code development or debugging. The executable name is simply **adpac.aix.dbx**.

aix_parallel This is the standard IBM RS-6000 AIX UNIX system compilation for the parallel version of the *ADPAC08* code. The standard APPL message passing library is incorporated, and therefore creation of this executable requires that a *make* has been issued in the APPL directory on the current machine. The *parallel* code may only be executed using the APPL *compute* function with a corresponding APPL *procdef* file. Prior to performing this compilation, the appropriate *make* command must be issued in the APPL directory to assemble the APPL library for the local machine. The executable name is simply **adpacp.aix**.

ncube This is the standard nCUBE 2 system compilation for the parallel version of the *ADPAC08* code. The nCUBE version of the APPL message passing library is incorporated, and therefore creation of this executable requires that a *make* has been issued in the APPL directory on the current machine. The *parallel* code may only be executed using the APPL *compute* function with a corresponding APPL *procdef* file. Prior to performing this compilation, the appropriate *make* command must be issued in the APPL directory to assemble the APPL library for the local machine. The executable name is simply **adpacp.ncube**.

ncube_csdb This option is identical to *ncube* above, except that the C-based scientific database libarary is linked instead of the standard scientific database libarary (which is thought to be slower). Prior to performing this compilation, the appropriate *make* command must be issued in the CSDB and APPL directories to assemble the CSDB and APPL libraries for the local machine. The executable name is simply **adpacp.ncube**.

At the completion of the compilation process on any system, an executable version of the code is written in the source directory (see Appendix A for an application of the compilation and execution processes for a sample test case).

3.5 *ADPAC08* Input/Output Files

In this section, the various input/output data files related to a calculation using the *ADPAC08* program are described. In order to understand the file naming convention, the concept of a case name must first be detailed. All files used in an *ADPAC08* calculation are named according to a standard naming convention of the form:

case.extension

where *case* is a unique, user-specifiable name identifying the geometry or flow condition being investigated, and *extension* is a name describing the type of file. The *case* name must be specified in the standard input file described below. A list and description of each of the files used or generated by *ADPAC08* is given in Table 3.1.

The standard input, standard output, boundary data, and convergence history files are stored in ASCII format. All other files utilize the Scientific DataBase Library (SDBLIB) [10] format. The mesh file and *PLOT3D* plot output files are compatible with the *PLOT3D* multiple grid, binary definition (see Sections 3.8 and 3.11 for a description and coding examples of the SDBLIB binary format). Files dealing with the parallel executaion of the *ADPAC08* code are described in Chapter 4.

The standard input and standard output files are directed at runtime using the standard UNIX redirection syntax as:

adpac < case.input > case.output

If a restart run is desired, the user must move the most current output restart file from

case.restart.new

to the default input restart file name

case.restart.old

each time the code is restarted. A more detailed description of the use and format of the *ADPAC08* files is presented in the sections which follow.

Table 3.1: Description of input/output files and UNIX-based filenames for *ADPAC08* Euler/Navier-Stokes solver

Name	Description
case.input	Standard input file
case.boundata	Block boundary definition file
case.output	Standard output file
case.mesh	Mesh file (*PLOT3D* compatible)
case.p3dabs	Final *PLOT3D* output file (absolute flow)
case.p3drel	Final *PLOT3D* output file (relative flow)
case.bf.#	2-D blockage/body force file for block number #
case.p3fr.#	Instantaneous *PLOT3D* interval output file (absolute flow). The frame number is given by #.
case.img.#	Instantaneous Silicon Graphics image file for graphics interactive display. The frame number is given by #.
case.restart.new	New restart file (output by code)
case.restart.old	Old restart file (used as input for restart runs)
case.converge	Solution residual convergence history file
case.sixpac	*SIXPAC* block subdivision file (parallel only)
Ncase.bacpac	*BACPAC* block reconstruction file (parallel only)
Ncase.blkproc	*ADPAC08* block/processor assignment file (parallel only)
procdef	APPL process description file (parallel only)

3.6 *ADPAC08* Standard Input File Description

The standard *ADPAC08* input file **case.input** contains the user-specifiable parameters which control the basic operation of the code during execution. These parameters tend to be less case dependent (as opposed to the boundary data file which is entirely case dependent). During code execution, the input file is read one line at a time as a character string, and each string is parsed sequentially to determine the specific program action in each case. The standard input file utilizes a keyword input format, such that any line which does not contain a recognizable keyword is treated as a comment line. Therefore, the user may place any number of comments in the file (so long as the line does not contain a keyword input string in the form described below), and code execution is unaltered. Comments may also be placed after the variable assigned to the keyword as long as there are one or more blanks separating the keyword value from the comment string. All input file lines are echoed to the standard output, and the program response to each line is listed when a specific action is taken.

All keyword input lines are given in the following format:

```
KEYWORD = Value      Comment String
```

where KEYWORD is one of the recognized keywords described below, and Value is the specific value to be assigned to that variable. The input line must contain the equals sign (=) with one or more blanks on **both** sides in order to be recognized. The Comment String must be separated by one or more blank spaces from the Value. Therefore, the lines

```
DIAM    = 10.000
DIAM                      =                     10.000
DIAM    = 10.000    This is the diameter.
```

are valid keyword input lines assigning the value 10.0 to the variable associated with the keyword DIAM. Conversely, the lines

```
DIAM= 10.000
DIAM =10.000
DIAM=10.000
```

are not recognizable keyword input lines (in spite of the presence of the keyword DIAM), because of the lack of proper placement of the blanks about the equals sign. The purpose for this restriction is to permit keyword variables in comment lines, and to help users to generate readable input files. All keyword values are either real numbers (which, in many cases, are converted to integers in the code) or character strings.

A sample *ADPAC08* standard input file containing a number of typical use keywords is listed below:

ADPAC08 Sample Standard Input File

```
----------------------------------------------------
ADPAC Input File
----------------------------------------------------
NASA 1.15 PRESSURE RATIO FAN - 2 BLADE ROWS
----------------------------------------------------
VARNAME  = VARIABLE VALUE     COMMENT
----------------------------------------------------
CASENAME   = nasa            The case name is "nasa"
FMULTI     = 3.0             Three mesh levels for multigrid
FSUBIT     = 1.0             1 subiteration on each coarse mesh level
FFULMG     = 1.0             Use "full" multigrid
FCOAG1     = 3.0             Start "full" multigrid on 3rd mesh level
FCOAG2     = 2.0             End "full" multigrid on 2nd mesh level
FITFMG     = 150.0           150 "full" multigrid iterations
RMACH      =     0.750000    Reference Mach Number
FINVVI     =     0.000000    Inviscid Flow
GAMMA      =     1.400000    Specific heat ratio
PREF       =  2116.220000    Reference Total Pressure (lbf/ft**2)
TREF       =   518.670000    Reference Total Temperature (Deg. R)
RGAS       =  1716.260010    Gas constant (ft-lbf/slug-deg R)
DIAM       =     9.000000    Reference diameter (ft.)
EPSX       =     1.500000    Residual smoothing in i direction
EPSY       =     1.500000    Residual smoothing in j direction
```

```
EPSZ       =    1.500000    Residual smoothing in k direction
VIS2       =    0.500000    Fine mesh 2nd order dissipation coefficient
VIS4       =    0.015625    Fine mesh 4th order dissipation coefficient
VISCG2     =    0.125000    Coarse mesh dissipation coefficient
CFL        =   -5.000000    Steady flow, CFL=5.0
FNCMAX     =  150.000000    150 iterations on fine mesh level
PRNO       =    0.700000    Gas Prandtl number = 0.7
PRTNO      =    0.900000    Turbulent Prandtl number = 0.9
FREST      =    0.000000    No restart file is read in
ADVR(1)    =   -2.780000    Advance ratio for block #1 is -2.78
ADVR(2)    =   -2.780000    Advance ratio for block #2 is -2.78
ADVR(3)    =    0.000000    Advance ratio for block #3 is  0.00
ADVR(4)    =    0.000000    Advance ratio for block #4 is  0.00
ENDINPUT
```

It is unnecessary to specify all possible keywords in every input file. The *ADPAC08* code is programmed with a default set of input variables such that the only input variable which must be present is the **CASENAME** (described below) which is used to assign input/output file names. A list and description of all input keywords and their default values are listed below. A quick reference to the input file keywords is provided in Appendix B.

ADPAC08 Standard Input File Keyword Description

ADVR(*NUM*)

(Default Value = 0.0)

```
ADVR(1) = 0.0
ADVR(2) = 0.0
ADVR(3) = 0.0
```

The **ADVR** keyword value determines the rotational speed (in terms of an advance ratio) of the mesh block number specified by the value *NUM*. Block rotational speeds are, by default, zero, unless either an **RPM** or an **ADVR** keyword are specified otherwise. The advance ratio is inherently tied to the freestream Mach number specified in the value associated with the keyword

RMACH. If the mesh has not been correctly non-dimensionalized, or if the value of **RMACH** is incorrect, it is possible that an incorrect value of rotational speed would be specified in the calculation.

BFFILE(*NUM*)

(Default Value = default_file_name)

```
BFFILE(1) = case.bf.b1
```

The **BFFILE** keyword value determines the name of the file used to read in the data for the blade blockage and body force source terms used to represent the effects of embedded blade rows in 2-D axisymmetric flow calculations. The file specified by **BFFILE** is used to describe the terms for the block number indicated by the value of *NUM*. Body force data files created by the *ADPAC08* program are named according to the file naming convention described in Section 3.5.

CASENAME

(No Default Value)

```
CASENAME = case
```

The **CASENAME** keyword value is used to set the case name which is used to define all input/output file names during an *ADPAC08* run (see Section 3.5 for details). The case name is limited to an 8 character string, and cannot contain embedded blanks. The case name has no default value, and as such, *all* input files *must* contain the **CASENAME** keyword.

CFL

(Default Value = -5.0)

```
CFL = -5.0
```

The **CFL** keyword defines the value of the time step multiplier used in the time-marching solver. The algorithm is sensitive to the sign of the value used for **CFL** in determining the manner in which the time-marching solver is applied. If **CFL** < 0.0, local time stepping is used (steady flow only) and each cell is advanced in time according to the local maximum allowable time step. If **CFL** > 0.0, then a time-accurate time-marching algorithm is applied, and the code calculates the smallest of all calculated maximum time steps and applies this value uniformly in the time-marching scheme at each cell. The absolute value of **CFL** is used as a multiplier for the time step (larger absolute values indicate larger time steps). A value of -5.0 is normally used for steady flow calculations, and values as high as 7.0 have been used successfully for time-accurate calculations. The value of **CFL** is also used implicitly in the eigenvalue scaling terms in the implicit residual smoothing algorithm, such that larger values of **CFL** imply increased residual smoothing (see the description of the implicit residual smoothing algorithm in the companion Final Report [1] and the description of **CFMAX**).

CFMAX

(Default Value = 2.5 (four stage scheme), 3.5 (five stage scheme))

```
CFMAX = 2.5
```

The **CFMAX** variable is used to define the maximum allowable time step multiplier for the explicit time-marching scheme without residual smoothing. This value is used in the implicit residual smoothing routine to adjust the smoothing coefficients for variations in time steps (see the Final Report [1]). Normally referred to as a CFL number, the variable **CFMAX** represents the maximum allowable CFL number for the time-marching scheme without residual smoothing, while the variable **CFL** represents the actual CFL number used in the calculation with residual smoothing. The ratio of **CFL** to **CFMAX** is used to adjust the amount of smoothing in the residual smoothing operator. Increasing **CFMAX** decreases the magnitude of the residual smoothing coefficients and therefore decreases the overall smoothing. Based on linear stability analysis, the four stage Runge-Kutta time-marching scheme permits a maximum CFL number of $2\sqrt{2}$. For simplicity, this value is normally approximated as 2.5 which provides an additional margin of safety. Under certain circumstances, it may be desirable to reduce **CFMAX** as low as 2.0 to aid convergence by artificially increasing the amount of residual smoothing. For the five stage scheme values of 3.0 to 3.5 are recommended.

CMUTSS, CMUTPS

(Default Value = 14.0)

```
CMUTSS = 14.0
CMUTPS = 14.0
```

The **CMUTSS, CMUTPS** keywords determine the ratio of local turbulent to laminar viscosity required to initiate transition for the point transition model in the *ADPAC08* body centered mesh turbulence model activated by the keyword **FTURBCHT**. This simplified transition model maintains laminar flow until the ratio of near wall turbulent viscosity to near wall laminar viscosity exceeds the value of **CMUTSS** or **CMUTPS** for the "suction side" and "pressure side", respectively, of the airfoil in question. The transition model parameters are illustrated in Figure 3.1. A ratio of 14.0 is recommended for all cases unless specific testing has indicated an alternate value.

DIAM

(Default Value = 1.0)

```
        DIAM = 1.0
```

The **DIAM** keyword is used as a dimensionalizing length scale for the mesh system for a given case. The *ADPAC08* code assumes that the mesh has been generated in a nondimensional fashion, and must be dimensionalized before execution. The value of the **DIAM** variable is used to convert the supposed nondimensional mesh coordinates into a dimensional length scale with units of feet. In other words, if the mesh has been generated using a length scale of inches, then the value of **DIAM** should be $\frac{1}{12}$, or 0.083333 in order to convert the mesh units to units of feet. If the mesh units are already in feet, then the value of **DIAM** should be simply 1.0. Many mesh generation systems for turbomachinery geometries nondimensionalize the mesh by a reference diameter determined from the turbomachinery geometry such that the maximum value of any radial coordinate in the mesh is 0.5. In this case, the value of **DIAM** should be the diameter of the turbomachine in feet used to nondimensionalize the mesh. Proper specification of the **DIAM** value is critical to achieve the correct flow Reynolds number and rotational speed for rotating geometries. Many problems can be traced to improper specification of the **DIAM** value and the user should take care to understand the use of this keyword. When in doubt, the user should remember the simple rule that the actual mesh units, when multiplied by the value of **DIAM** should result in physical lengths expressed in feet.

ENDINPUT

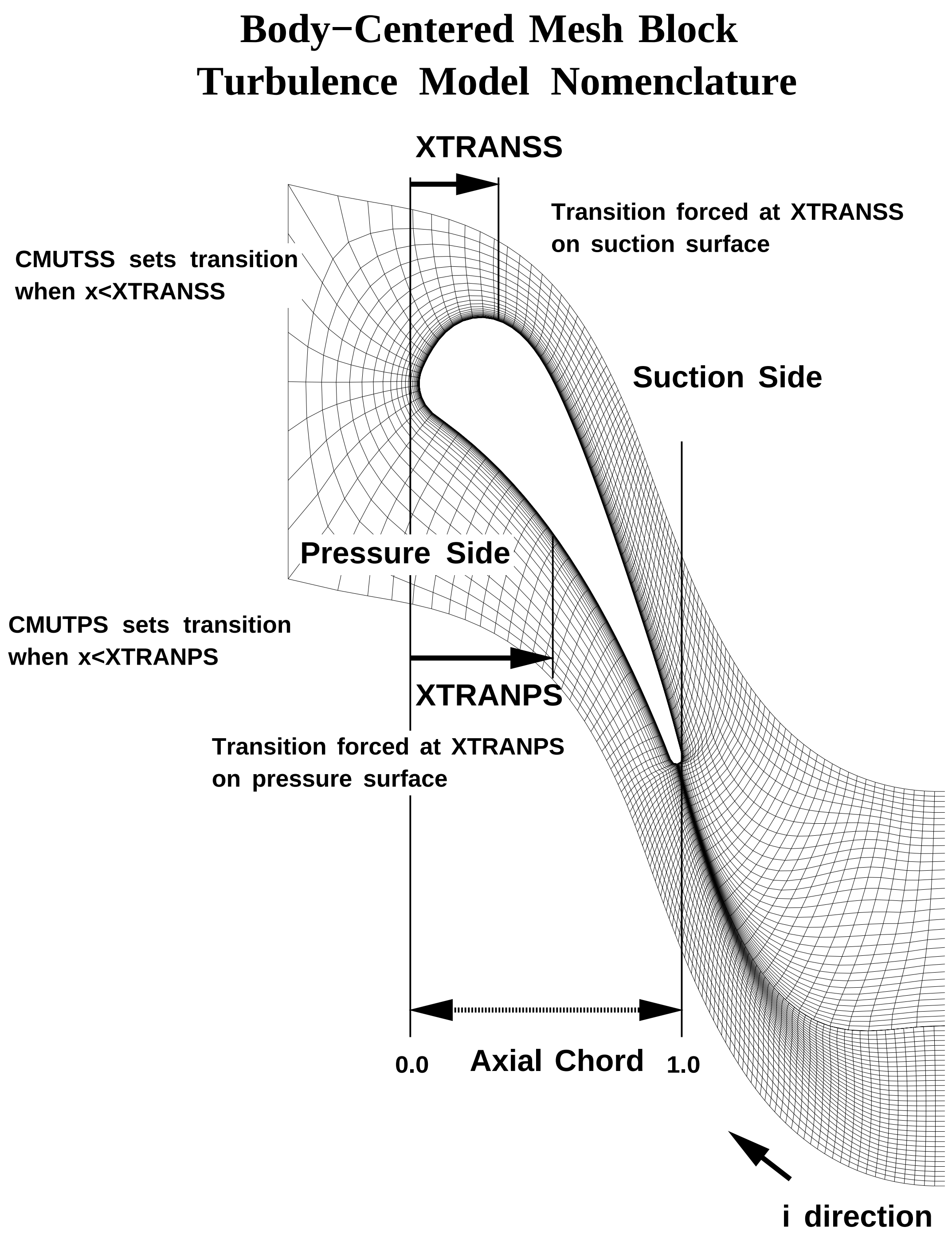


Figure 3.1: *ADPAC08* Body-Centered Mesh Turbulence Model Nomenclature Summary

```
ENDINPUT
```

When the *ADPAC08* program encounters the keyword **ENDINPUT**, the parser which searches each line for a valid input keyword string is terminated, and no additional input file lines are parsed for input keyword values. Any lines following the **ENDINPUT** statement are ignored, except when the graphics display system is in effect across a network, in which case the statements following the **ENDINPUT** statement must contain two blank lines and the Internet network address of the destination display device (see Chapter 9 for a description of the Interactive Graphics Display option).

EPSTOT

(Default Value = 0.1)

```
EPSTOT = 0.1
```

The **EPSTOT** keyword determines the value of the smoothing coefficient employed in the post multigrid smoothing algorithm described by the trigger **FTOTSM**. This coefficient is only used when **FTOTSM** = 1.0. The value of the coefficient may be any positive number, but for most circumstances, a value between 0.0 and 0.25 is suggested (larger values imply more smoothing).

EPSX, EPSY, EPSZ

(Default Value = 1.0)

```
EPSX = 1.0
EPSY = 1.0
EPSZ = 1.0
```

The **EPSX, EPSY, EPSZ** keywords set the value of the implicit residual smoothing coefficient multipliers in the i, j, and k coordinate directions, respectively. The values of **EPSX, EPSY,** and **EPSZ** are used as simple multipliers for the residual smoothing coefficients calculated by the eigenvalue scaling residual smoothing scheme described in the Final Report [1]. If **EPSX, EPSY** or **EPSZ** = 0.0, then no smoothing is applied for the given coordinate direction. The user should be aware that the keyword variable **FRESID** controls the global application of residual smoothing in the solution algorithm, and in the case where **FRESID**=0.0 (residual smoothing disabled), the **EPSX, EPSY, EPSZ** have no impact on the solution. The default value for the coefficient multipliers is 1.0. Any value larger than 1.0 simply implies excess smoothing and may be useful for cases with poor convergence or undesirable mesh quality. If a value larger than 3.0 is required to stabilize a solution, this generally indicates some sort of problem in the calculation (such as poor mesh aspect ratio, bad boundary specification, etc.), or might suggest that **FRESID** has been set to 0.0. Values less than 1.0 will likely cause code instabilities for values of **CFL** greater than 2.0.

FBCONF

(Default Value = 99999.0)

```
FBCONF = 99999.0
```

The **FBCONF** keyword assigns a trigger which determines the iteration number at which the boundary conditions are frozen. This trigger was added for those cases where convergence is apparently hindered by "noise" from the boundary conditions. Caution must be exercised when using the **FBCONF** variable due to the fact that the *ADPAC08* code could ultimately diverge when all of the boundary conditions are frozen.

FCARB(*NUM*)

(Default Values = 0.0)

```
FCARB(1) = 1.0
FCARB(2) = 0.0
FCARB(3) = 0.0
FCARB(4) = 0.0
FCARB(5) = 0.0
```

The keyword **FCARB(***NUM***)** sets a block specific trigger for the mesh block number specified by *NUM* which determines, on a block by block basis, whether the Cartesian (**FCARB(***NUM***)** = 1.0) or the cylindrical (**FCARB(***NUM***)** = 0.0) solution algorithm is employed by that block. The *ADPAC08* code permits mixed cylindrical and Cartesian solution blocks in a single calculation. While the variable **FCART** may be used to set the global value of mesh blocks for either cylindrical or Cartesian solution status, the variable **FCARB(***NUM***)** may be utilized to set specific blocks one way or the other. It must be noted that the variable **FCARB(***NUM***)** will always override the status implied by **FCART**. At present, the only boundary condition which permits interblock communication between mixed cylindrical and Cartesian blocks is **BCPRR** (see Section 3.7).

FCART

(Default Value = 0.0)

```
FCART = 0.0
```

The **FCART** keyword assigns a trigger which controls the cylindrical/Cartesian coordinate system solution scheme in the the *ADPAC08* code. If **FCART** = 0.0, then all blocks (except those specifically altered by the **FCARB** input variable) are treated as cylindrical coordinate system blocks. If **FCART** = 1.0, then all blocks (except those specifically altered by the **FCARB** input variable) are treated as Cartesian coordinate system blocks.

FCOAG1

(Default Value = 1.0)

```
FCOAG1 = 1.0
```

The **FCOAG1** keyword specifies the initial, or coarsest coarse mesh level upon which the "full" multigrid calculation is initially applied (for additional details, see the description of **FFULMG**). When multiple coarse mesh levels are available for processing, it is occasionally useful to specify the initial coarse mesh level in the "full" multigrid sequence in order to avoid wasted computations on lower mesh levels. Typically, **FCOAG1** is set to **FMULTI** (start "full" multigrid on coarsest mesh level). In some cases (when **FMULTI** is larger than 3.0) it may be advisable to set **FCOAG1** to 3.0, and avoid useless processing on coarser meshes during the "full" multigrid startup process. A flowchart of the *ADPAC08* iteration and multigrid control algorithm is given in Figure 3.2. An example is given in the description of **FCOAG2**.

FCOAG2

(Default Value = 1.0)

```
FCOAG2 = 1.0
```

The **FCOAG2** keyword specifies the final, or finest coarse mesh level upon which the “full” multigrid calculation is applied (for additional details, see the description of **FFULMG**). When multiple coarse mesh levels are available for processing, it is occasionally useful to specify the final coarse mesh level in the “full” multigrid sequence in order to examine the flowfield without actually performing any calculations on the fine mesh. For example, the combination

```
FMULTI = 3.0
FCOAG1 = 3.0
FCOAG2 = 3.0
FNCMAX = 10.0
FITFMG = 100.0
```

would direct a “full” multigrid startup of 100 iterations on mesh level 3, and since **FCOAG2**=3.0, the “full” multigrid sequence is ended at this mesh level. The solution is then interpolated to the fine mesh, and then 10 fine mesh iterations using 3 levels of multigrid would be performed. Typically, **FCOAG1** is set to 2.0, which indicates that the “full” multigrid startup procedure utilizes all mesh levels from **FCOAG1** to 2 before starting any processing on the fine mesh. A flowchart of the *ADPAC08* iteration and multigrid control algorithm is given in Figure 3.2.

FDEBUG(*NUM*)

(Default Values = 0.0)

```
FDEBUG(1) = 0.0
FDEBUG(2) = 0.0
FDEBUG(3) = 3.0
FDEBUG(4) = 0.0
FDEBUG(5) = 0.0
```

The keyword **FDEBUG(***TYPE***)** defines a block number for the debug output type specified by *TYPE* which determines, on a type by type basis, whether debug output from the *ADPAC08* run is printed to the standard output. When enabled, this variable will generate an extreme amount of output and should therefore be used only in a controlled debugging environment. The value of the variable FDEBUG(*TYPE*) determines for which blocks the particular type of output is enabled. The following debug output types are currently supported:

FDEBUG(1)	Print the input (Cartesian) mesh points
FDEBUG(2)	Print the (converted) cylindrical mesh points
FDEBUG(3)	Print the cell face areas
FDEBUG(4)	Print the cell volumes
FDEBUG(5)	Print the cell flow data
FDEBUG(6)	Print the cell time steps
FDEBUG(7)	Print the cell convective fluxes
FDEBUG(8)	Prin the cell dissipative fluxes
FDEBUG(9)	Print the cell diffusive fluxes
FDEBUG(10)	Print the cell implicit residual smoothing data

FFILT

(Default Value = 1.0)

```
FFILT = 1.0
```

The **FFILT** keyword assigns a trigger which determines directly whether the added dissipation routines are called during the time-marching process. If **FFILT = 0.0**, then no added dissipation is calculated. It is also possible to turn off the added dissipation by setting the values of **VIS2** and **VIS4** to 0.0; however, the use of **FFILT** avoids the calculation of the dissipation terms entirely. It is unlikely that any value other than 1.0 is required except for code debugging purposes.

FFULMG

(Default Value = 0.0)

```
          FFULMG = 0.0
```

The **FFULMG** keyword assigns a trigger which determines whether the "full" multigrid solution procedure is applied or whether the standard multigrid procedure is used. The use of "full" multigrid is advisable (but not required) when a new calculation is being started as a means of rapidly generating a better initial guess for the final flowfield. If the solution is being restarted from a previous calculation (FREST=1.0), it is usually advisable to set **FFULMG** to 0.0 to avoid destroying the initial data read from the restart file (a warning message is issued when this combination is specified). A flowchart of the *ADPAC08* iteration and multigrid control algorithm is given in Figure 3.2.

FGRAFINT

(Default Value = 1.0)

```
          FGRAFINT = 1.0
```

The **FGRAFINT** keyword determines the number of iterations between flowfield display updates for the *ADPAC08* real time graphics display system. This option is only valid when **FGRAFIX** = 1.0, and is subject to a number of other restrictions for the graphics display system (see the description of input keywords **FGRAFIX** and **FIMGSAV**, and the description of the graphics display system, Chapter 9). The default value for **FGRAFINT** is 1.0, which indicates that the graphics display will be updated every iteration. This can cause excessive computational and network overhead, and the user should be aware of the potential problems when using the graphics display features.

FGRAFIX

(Default Value = 0.0)

```
          FGRAFIX = 0.0
```

The **FGRAFIX** keyword sets a trigger which controls the generation of the real time interactive graphics display in the *ADPAC08* program. A value of **FGRAFIX** = 1.0 indicates that the interactive graphics display facility is desired, while **FGRAFIX** = 0.0 turns this option off. When functional, the graphics screen is updated with the latest available flow data every **FGRAFINT** iteration. Graphics images can be automatically captured on specific computer hardware every **FIMGSAV** iterations as a means of creating flowfield animations (see Graphics Display, Chapter 9).In order for the graphics display to work, the code must be compiled with either the *graphics* or *pfagraphics* Makefile option (see Section 3.4 for a description of the Makefile and the *ADPAC08* code compilation process). There are also specific machine requirements for this option to work as well (see the section on Graphics Display, Chapter 9). The generation of interactive, real time graphics images increases the overall computational cost, and can cause network overloading in some cases due to the transmission of graphics information.

FIMGINT

(Default Value = 99999.0)

```
          FIMGINT = 99999.0
```

The **FIMGINT** keyword determines the number of iterations between flowfield graphics display image capturing available on Silicon Graphics computers for the *ADPAC08* real time graphics display system. This option is only valid when **FGRAFIX** = 1.0, and **FIMGSAV** = 1.0, and is subject to a number of other restrictions for the graphics display system (see the description of input keywords **FGRAFIX** and **FGRAFINT**, and the description of the graphics display system, Chapter 9). The default value for **FIMGINT** is 99999.0, which indicates that a screen image will be saved every 99999 iterations. This large value was chosen to prohibit accidental image capturing, which can quickly fill up a large amount of disk storage. The graphics display system can cause excessive computational and network overhead, and the user should be aware of the potential problems when using this feature of the *ADPAC08* code.

FIMGSAVE

(Default Value = 0.0)

```
          FIMGSAV = 0.0
```

The **FIMGSAV** keyword sets a trigger which controls the Silicon Graphics computer screen image capturing facility of the real time interactive graphics display in the *ADPAC08* program. A value of **FIMGSAV** = 1.0 indicates that the graphics image capturing facility is desired, while **FIMGSAV** = 0.0 turns this option off. When the interactive graphics display option has been enabled (see details for input keywords **FGRAFIX, FGRAFINT**) the graphics screen is updated with the latest available flow data every **FGRAFINT** iteration. When the image capturing facility is enabled, these graphics images can be automatically captured on specific computer hardware every **FIMGINT** iterations as a means of creating flowfield animations (see Graphics Display, Chapter 9).In order for the graphics display image capturing facility to work, the code must be compiled with either the *graphics*, or *pfagraphics* Makefile option (see Section 3.4 for a description of the Makefile and the *ADPAC08* code compilation process). There are also specific machine requirements for this option to work as well (see the section on Graphics Display, Chapter 9). The generation of interactive, real time graphics images increases the overall computational cost, and can cause network overloading in some cases due to the transmission of graphics information. The capturing of many screen images will also require a large amount of file storage space (see Section 3.5 for a description of the image capturing file naming convention).

FINVVI

(Default Value = 0.0)

```
          FINVVI = 0.0
```

The **FINVVI** keyword is a simple trigger to determine whether the solution mode is for inviscid flow (**FINVVI** = 0.0) or for viscous flow (**FINVVI** = 1.0). This trigger controls whether the viscous stress flux contributions are calculated during the time-marching process. This does not affect the application of boundary conditions, as this is completely controlled by the specifications in the boundary data file (see Section 3.7). As such, it is possible to run viscous boundary conditions in an inviscid flow solution, and inviscid boundary conditions in a viscous flow solution.

FITCHK

(Default Value = 100.0)

```
          FITCHK = 100.0
```

The **FITCHK** keyword controls the number of iterations between job checkpointing in the *ADPAC08* program. Job checkpointing refers to the process of periodically saving the flowfield information to minimize the loss of data in the event that the job does not terminate normally. As a safety feature, the *ADPAC08* program writes out an updated restart file every **FITCHK** iterations in case the job stops before the final restart file output procedures are completed. It is not necessary to write out intermediate restart files, but this is considered a good precaution against unexpected problems such as computer failures, or system administration quotas. A good interval for checkpointing is 100 iterations (**FITCHK** = 100.0). The intermediate restart files, as well as the final restart file, are all written to the same file name, and therefore previous checkpoints cannot be retrieved when the file is overwritten (see Section 3.5 for restart file naming conventions). Job checkpointing only applies to the iterative cycles involving the fine mesh, and does not apply to the coarse mesh iterations calculated during a "full" multigrid startup (see **FFULMG**).

<u>FITFMG</u>

(Default Value = 100.0)

```
FITFMG = 100.0
```

The **FITFMG** keyword dictates the number of iterations to be performed on each of the coarse mesh levels during a “full” multigrid startup sequence (see the description of **FFULMG**). Typically, the startup sequence is used only to generate a reasonable initial guess for the fine mesh, so the value of **FITFMG** is kept relatively low ($\approx$ 100). The function of the keyword **FITFMG** is illustrated graphically in Figure 3.2.

<u>FMASSAVG</u>

(Default Value = 3.0)

```
FMASSAVG = 3.0
```

The **FMASSAVG** keyword describes a trigger which determines the type of mass averaging used in the *ADPAC08* code for various boundary conditions. In particular, the **MBCAVG** boundary condition which imposes a circumferential mixing plane for simplified representation of multistage turbomachinery blade row flows is the most commonly affected roputine. A value of **FMASSAVG** = 0.0 implies that an algebraic average is to be used in the averaging operator. A value of **FMASSAVG** = 1.0 implies that an area average is to be used in the averaging operator. A value of **FMASSAVG** = 2.0 implies that a true mass-weighted average is to be used in the averaging operator. Values of 2.0 and 3.0 are recommended, as the algebraic average introduces a fair amount of error.

FMULTI

(Default Value = 1.0)

```
          FMULTI = 1.0
```

The **FMULTI** keyword assigns the number of multigrid levels to be used during the calculation (for a description of the multigrid algorithm, see Reference [1]). The code will analyze the dimensions of the fine mesh to determine whether it can be properly subdivided according to the number of multigrid levels requested. If **FMULTI** $\leq$ 1.0, then the number of multigrid levels is set to 1, and the calculation is performed on the finest mesh only without multigrid acceleration. For unsteady flows, multigrid is not valid, and **FMULTI** should be set to 1.0. A flowchart of the *ADPAC08* iteration and multigrid control algorithm is given in Figure 3.2.

FNCMAX

(Default Value = 100.0)

```
          FNCMAX = 100.0
```

The **FNCMAX** keyword controls the total number of iterations for a non-multigrid calculation (**FMULTI** $\leq$ 1.0), or the number of global iterations on the finest mesh for a multigrid calculation (**FMULTI** $>$ 1.0). The total number of iterations performed on all meshes for a multigrid run is controlled by a combination of **FNCMAX**, **FMULTI**, **FCOAG1**, **FCOAG2**, **FFULMG**, **FITFMG**, and **FSUBIT**. For example, the values

```
          FNCMAX = 200.0
          FMULTI = 1.0
          FITFMG = 0.0
          FFULMG = 0.0
          FSUBIT = 0.0
```

would prescribe 200 iterations of a non-multigrid run (only the fine mesh is used). The values

```
FNCMAX = 200.0
FMULTI = 3.0
FITFMG = 0.0
FFULMG = 0.0
FSUBIT = 1.0
```

would prescribe 200 multigrid iterations using 3 mesh levels (but still only 200 global iterations, where each iteration involves a single subiteration on each of 3 mesh levels). And finally, the values

```
FNCMAX = 200.0
FMULTI = 3.0
FITFMG = 50.0
FFULMG = 1.0
FSUBIT = 1.0
FCOAG1 = 3
FCOAG2 = 2
```

would prescribe an initial pass of 50 iterations on the third mesh level, followed by 50 multigrid iterations on the second mesh level, and finally 200 global multigrid iterations on the finest mesh level. See the descriptions of the variables **FNCMAX**, **FMULTI**, **FCOAG1**, **FCOAG2**, **FFULMG**, **FITFMG**, and **FSUBIT** for further details. A flowchart of the *ADPAC08* iteration and multigrid control algorithm is given in Figure 3.2.

FRDMUL

(Default Value = 0.0)

```
FRDMUL = 0.0
```

The **FRDMUL** keyword determines whether boundary condition data for the coarse mesh levels of a multigrid run are generated from the fine mesh boundary conditions specified in the *ADPAC08* boundary data file (**FRDMUL** = 0.0), or whether the coarse mesh boundary specifications are actually read in from the boundary data file (**FRDMUL** = 1.0). In most cases,

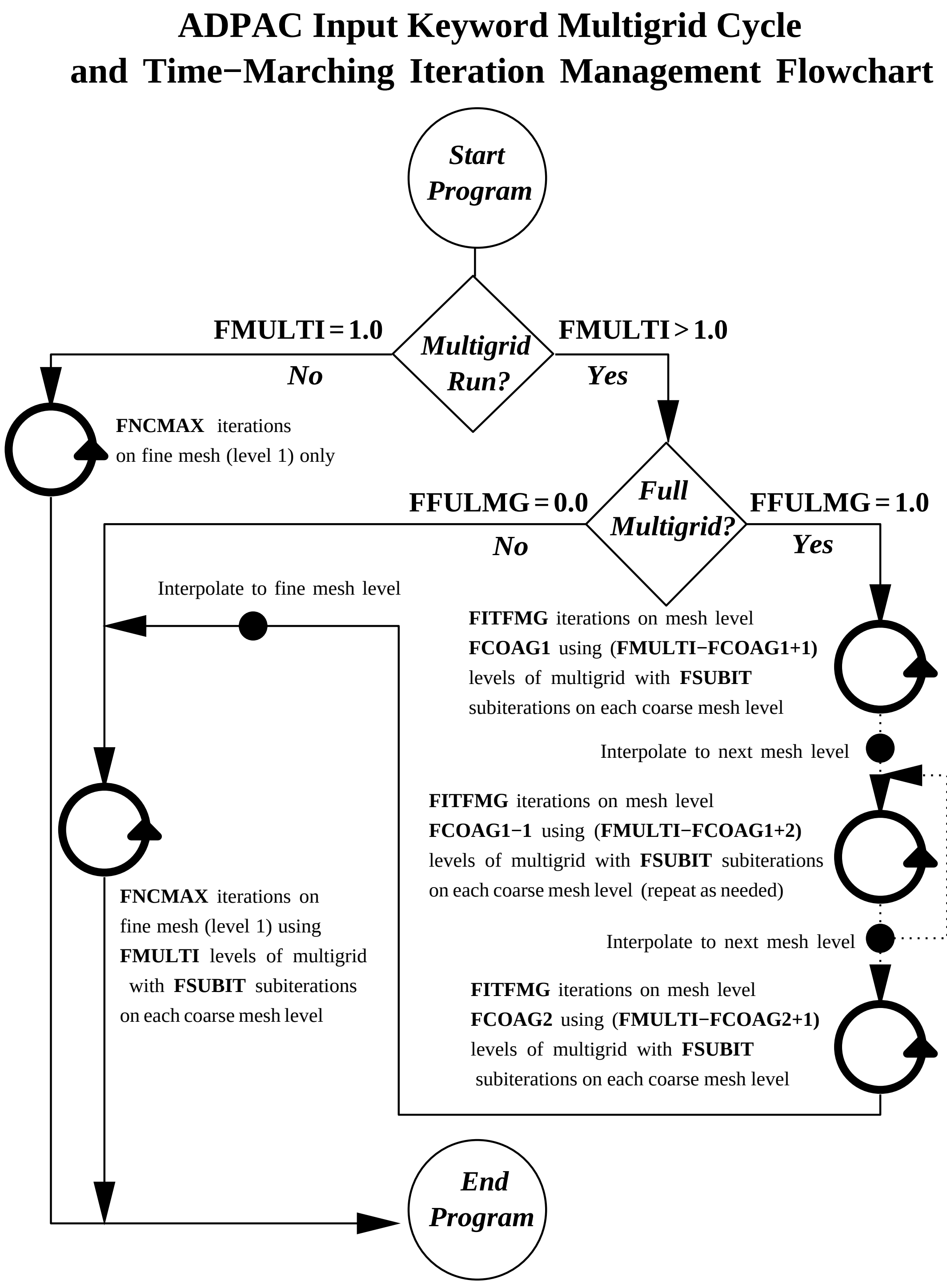

ADPAC Input Keyword Multigrid Cycle
and Time−Marching Iteration Management Flowchart
Start Program
Multigrid Run?
FMULTI = 1.0
No
FMULTI > 1.0
Yes
FNCMAX iterations on fine mesh (level 1) only
Full Multigrid?
FFULMG = 0.0
No
FFULMG = 1.0
Yes
Interpolate to fine mesh level
FITFMG iterations on mesh level FCOAG1 using (FMULTI−FCOAG1+1) levels of multigrid with FSUBIT subiterations on each coarse mesh level
Interpolate to next mesh level
FITFMG iterations on mesh level FCOAG1−1 using (FMULTI−FCOAG1+2) levels of multigrid with FSUBIT subiterations on each coarse mesh level (repeat as needed)
Interpolate to next mesh level
FITFMG iterations on mesh level FCOAG2 using (FMULTI−FCOAG2+1) levels of multigrid with FSUBIT subiterations on each coarse mesh level
FNCMAX iterations on fine mesh (level 1) using FMULTI levels of multigrid with FSUBIT subiterations on each coarse mesh level
End Program

Figure 3.2: *ADPAC08* Input Keyword Multigrid Cycle and Time-Marching Iteration Management Flowchart

FRDMUL should be set to 0.0, and the program will determine the equivalent coarse mesh boundary conditions from the fine mesh specifications. For the purposes of code debugging, or to permit multigrid calculation on a mesh which does not possess perfect “multigrid” boundary segments (a boundary condition for the fine mesh does not begin or end at a mesh index which is compatible with the multigrid sequence), it is possible to “fool” the program into running multigrid by artificially specifying an equivalent coarse mesh boundary condition.

FRESID

(Default Value = 1.0)

```
FRESID = 1.0
```

The **FRESID** keyword assigns a trigger which determines directly whether the implicit residual smoothing routines are called during the time-marching process. If **FRESID = 0.0**, then no residual smoothing is applied. It is also possible to turn off the residual smoothing by setting the values of **EPSX**, **EPSY**, and **EPSZ** to 0.0; however, the use of **FRESID** avoids the calculation of the smoothed residuals entirely. It is unlikely that any value other than 1.0 is required except for code debugging purposes, or for calculations involving **CFL**$\leq$ 2.0.

FREST

(Default Value = 0.0)

```
FREST = 0.0
```

The **FREST** keyword assigns a trigger which controls the restart characteristics of the *ADPAC08* code. If **FREST** = 0.0, then no restart file is used, and the flow variables are initialized according to the scheme described by the input keyword **RMACH**. If **FREST** = 1.0, then the code attempts to open a restart file (*case.restart.old*) described by the file naming convention (see Section 3.5), and the flow variables are initialized by the values given in the restart file. Restarting a calculation from a previous calculation is often useful for breaking up large calculations into smaller computational pieces, and may also provide faster convergence for cases which involve minor changes to a previous calculation.

FSAVE

(Default Value = 1.0)

```
FSAVE = 1.0
```

The **FSAVE** keyword assigns a trigger which controls the restart file output characteristics of the *ADPAC08* code. If **FSAVE** = 0.0, then no restart file is written at the end of an *ADPAC08* run. If **FSAVE** = 1.0, then the code attempts to open a restart file (*case.restart.new*) described by the file naming convention (see Section 3.5), and the flow variables are written to the restart file for future processing. Restarting a calculation from a previous calculation is often useful for breaking up large calculations into smaller computational pieces, and may also provide faster convergence for cases which involve minor changes to a previous calculation. The recommended procedure is to *always* write a restart file.

FSOLVE

(Default Value = 1.0)

```
FSOLVE = 1.0
```

The **FSOLVE** keyword assigns a trigger which determines which type of time-marching strategy is employed on both fine and coarse meshes. For **FSOLVE** = 0.0, the standard 4-stage Runge Kutta time-marching scheme is used with a single added dissipation evaluation, and implicit residual smoothing at alternating stages. For **FSOLVE** = 1.0, a modified 4-stage Runge Kutta time-marching scheme is used with a two evaluations of the added dissipation, and implicit residual smoothing at every stage. For **FSOLVE** = 2.0, a 5-stage Runge Kutta time-marching scheme is used with three weighted added dissipation evaluations, and implicit residual smoothing at every stage. **FSOLVE** = 1.0 is the recommended value, although the other schemes may provide improved convergence at a somewhat different computational cost.

FSUBIT

(Default Value = 1.0)

```
FSUBIT = 1.0
```

The **FSUBIT** keyword determines the number of subiterations performed on coarse meshes during the coarse mesh portion of the multigrid cycle. As such, this variable is actually only used when **FMULTI** > 1.0. Additional subiterations on coarse meshes during the multigrid cycle can often improve convergence, at the expense of some additional computational work. The number of subiterations specified by **FSUBIT** is applied at each coarse mesh level during the multigrid calculations process. A value of 1.0, 2.0, or 3.0 is typically best. A complete description of the multigrid calculation process is given in the Final Report [1]. A flowchart of the *ADPAC08* iteration and multigrid control algorithm is given in Figure 3.2.

FTIMEI

(Default Value = 1.0)

```
          FTIMEI = 1.0
```

The **FTIMEI** keyword assigns a trigger which determines the number of iterations between time step evaluations. For best results, this should be 1.0, which implies that the time step is re-evaluated at every iteration. However, this value can be increased (< 10) to reduce CPU time by reevaluating the time step every **FTIMEI** iterations instead (at the possible expense of irregular convergence).

FTIMERM

(Default Value = 0.0)

```
          FTIMERM = 0.0
```

The **FTIMERM** keyword is utilized to control CPU quota during a run of the *ADPAC08* code. The effect of this variable is different during execution on a Cray computer and a UNIX workstation. On the Cray, for jobs running under the Network Queuing System (NQS), any nonzero value for **FTIMERM** directs the code to determine how much CPU time remains allocated to the current job during each time-marching iteration, and the *ADPAC08* code estimates how much of that CPU time is required to normally shut down the current job. If the time remaining to the allocation is indicated by $TIME$, and if the time required to shutdown is $SHUT$, then the code will evaluate the expression

$$TIME - SHUT + FTIMERM$$

where each term is in CPU seconds. If this expression is less than 0.0, then the code will halt the time marching process and attempt to shut down so the various output files can be written prior to termination by NQS due to CPU quota. Note that if **FTIMERM** is a negative number, then the code will shut down "early" in case additional programs must run under a given NQS run. On a UNIX workstation, NQS is usually not available, and in this case, the code keeps track of accumulated CPU time and terminates normal job processing when the accumulated CPU time exceeds the value of **FTIMERM**.

FTOTSM

(Default Value = 0.0)

```
FTOTSM = 0.0
```

The **FTOTSM** keyword is used to trigger the post multigrid smoothing algorithm. In this scheme, the residual corrections from the multigrid process are combined with the fine mesh residuals and are smoothed globally using a simple constant coefficient version of the implicit residual smoothing algorithm. The smoothing coefficient is determined by the value of the input keyword variable **EPSTOT**. The scheme is disabled when **FTOTSM** has a value of 0.0, and is employed when **FTOTSM** has a value of 1.0. This scheme has been found to aid stability, but can actually hinder convergence in some cases.

FTURBB

(Default Value = 10.0)

```
FTURBB = 10.0
```

The **FTURBB** keyword assigns a trigger which determines the number of iterations before the turbulence model is activated. For laminar flow, set **FTURBB** to a very large number (**FTURBB** > **FNCMAX** + (**FMULTI**-1) * **FITFMG**) * **FFULMG**) so the turbulence model is never called. For turbulent flow, the value should be large enough (e.g., ≥ 10) to ensure that the solution has developed adequately enough to permit stable implementation of the turbulence model (i.e., the flowfield should at least exhibit the gross characteristics (correct flow direction, some boundary layer development) of the expected final flow before the turbulence model is activated).

FTURBCHT(*NUM*)

(Default Values = 0.0)

```
FTURBCHT(1) = 1.0
FTURBCHT(2) = 0.0
FTURBCHT(3) = 0.0
FTURBCHT(4) = 0.0
FTURBCHT(5) = 0.0
```

The keyword **FTURBCHT(***NUM***)** sets a block specific trigger for the mesh block number specified by the value *NUM* to enable the body-centered mesh turbulence model described in Figure 3.1. If **FTURBCHT(***NUM***)** is set to 0.0, the standard turbulence model is used for the block specified by *NUM*. If **FTURBCHT(***NUM***)** is set to 1.0, then the special transition and body centered turbulence model is used for the block specified by *NUM*. The body-centered turbulence model locates the airfoil leading and trailing edges, and utilizes axial chord notation in conjunction with the input variables **XTRANSS, XTRANPS** and **CMUTSS, CMUTPS** to determine the natural transition point on the airfoil. This turbulence model was developed during an analysis of surface heat transfer (where transition plays a critical role) on a turbine vane cascade using a C-type mesh. The use of this model is recommended whenever the mesh topology is compatible with the scheme illustrated in Figure 3.1.

FTURBF

(Default Value = 99999.0)

```
FTURBF = 99999.0
```

The **FTURBF** keyword assigns a trigger which determines the iteration number at which the turbulence model is frozen. This trigger was added for those cases where convergence is apparently hindered by "noise" from the turbulence model. Caution must be exercised when using the **FTURBF** variable due to the fact that the *ADPAC08* restart file does not contain any turbulent viscosity data. If the *ADPAC08* code is restarted from a turbulent flow solution when the value of **FTURBF** is less than the current iteration level, no turbulent quantities will be generated and the flow will exhibit laminar flow characteristics.

FTURBI

(Default Value = 1.0)

```
FTURBI = 0.0
```

The **FTURBI** keyword assigns a trigger which determines the number of iterations between turbulence model evaluations. For best results, this should be 1.0, which implies that the turbulence parameters are reevaluated at every iteration. However, this value can be increased (< 10) to reduce CPU time by reevaluating the turbulence quantities every **FTURBI** iterations instead (at the possible expense of irregular convergence).

FUNINT

(Default Value = 99999.0)

```
FUNINT = 99999.0
```

The **FUNINT** keyword is used to determine the number of iterations between instantaneous *PLOT3D* absolute flow file output. For a time-dependent calculation, it is often desirable to print out data at several intervals during the course of the solution to examine the time-dependent nature of the flow. The *ADPAC08* program provides a mechanism by which a *PLOT3D* format flow file can be printed at a fixed iteration interval (the interval defined by the value of **FUNINT**) as a means of extracting time-dependent data during a calculation. For steady flow calculations, it is normally desirable to set **FUNINT** to a very large number, and simply use the final output *PLOT3D* format files if needed. For unsteady flow calculations, the value of **FUNINT** can be highly case dependent, and some numerical experimentation may be required to prevent excessive output, or a deficiency in data. The file naming convention for the unsteady output files is given in Section 3.5.

FUPWIND

(Default Value = 0.0)

```
FUPWIND = 0.0
```

The **FUPWIND** keyword is a simple trigger to activate the upwind differencing scheme (on=1.0, off=0.0) available for the 2-D mesh block solver in the *ADPAC08* code. The upwind differencing scheme is a first order scheme available for experimentation only, and is not a recommended solution technique for real flow calculations at this point.

FVTSFAC

(Default Value = 2.5)

```
FVTSFAC = 2.5
```

The **FVTSFAC** keyword determines the value of the viscous time step evaluation factor used to stabilize the time-marching solution for viscous flows. This factor is used to magnify the importance of the diffusion-related contributions to the time step evaluation (larger values suggest larger restrictions due to diffusion related parameters). This factor is particularly useful for meshes with rapid changes in grid spacing, and the default value of 2.5 was prescribed somewhat arbitrarily following numerical experimentation. It is unlikely that this value needs modification for most cases.

FWALLF

(Default Value = 1.0)

```
FWALLF = 1.0
```

The **FWALLF** keyword is used to trigger the use of wall functions in the *ADPAC08* turbulence model. Wall functions are normally desirable for meshes which are not highly clustered near solid surfaces. The *ADPAC08* code can determine when the wall function model is necessary and will automatically disable the wall function model (even if **FWALLF** is enabled) in favor of the standard turbulence model wall treatment for meshes with acceptable near wall spacing (roughly 0.0001 times airfoil chord for turbomachinery applications). The wall function model is not recommended for applications involving significant heat transfer.

GAMMA

(Default Value = 1.4)

```
        GAMMA = 1.4
```

The **GAMMA** keyword sets the value for the gas specific heat ratio. For most cases involving air at moderate pressures and temperatures, a value of 1.4 is adequate. For cases involving combustion products, this value may be quite different, and should be considered appropriately. Extreme care must be taken when post-processing a calculation which is based on a value of **GAMMA** other than 1.4 as many post processors use an assumed value of the specific heat ratio equal to 1.4 (*PLOT3D* is a common example). It should be mentioned that the present version of the code does not permit user specification of the fluid viscosity, as the formula for air is hardwired into the code.

P3DPRT

(Default Value = 1.0)

```
        P3DPRT = 1.0
```

The **P3DPRT** keyword assigns a trigger which determines whether *PLOT3D* format output files are written at the end of a calculation. A value of **P3DPRT** = 1.0 indicates that the output files should be written. Conversely, a value of **P3DPRT** = 0.0 indicates that the *PLOT3D* format output files should not be written. The *PLOT3D* output files (see Section 3.5 for file naming conventions for output files) are useful for graphically examining the predicted flow quantities using widely available plotting software such as *PLOT3D*, *FAST*, *SURF*, etc. Occasionally, however, due to disk space limitations or simply to speed up execution, it may be desirable to eliminate this output feature, and therefore **P3DPRT** can be used to control this output.

PREF

(Default Value = 2116.22)

```
        PREF = 2116.22
```

The **PREF** keyword sets the dimensional value (in pounds force per square foot) of the reference total pressure used to nondimensionalize the flowfield. For viscous flows, this value must be accurately specified in order to properly set the nondimensional flow viscosity, (and hence, the Reynolds number). For inviscid flow predictions, this value has no real significance because of the similarity of inviscid flows with Mach number. It is very important to choose an average representative value for this variable, such that the nondimensional total pressure at any point in the flow is near a value of 1.0. An extended discussion on the reason for this choice is given in the description of **RMACH**.

PRNO

(Default Value = 0.7)

```
PRNO = 0.7
```

The **PRNO** keyword assigns the value of the gas Prandtl number. For air (and many other gases) at moderate pressures and temperatures, a value of 0.7 is appropriate.

PRTNO

(Default Value = 0.9)

```
PRTNO = 0.9
```

The **PRTNO** keyword assigns the value of the gas turbulent Prandtl number. The turbulence model in *ADPAC08* determines the turbulent thermal conductivity via a turbulent Prandtl number and the calculated turbulent viscosity (see the Final Report [1]). The recommended value is 0.9.

RGAS

(Default Value = 1716.26)

```
RGAS = 1716.26
```

The **RGAS** keyword sets the dimensional value (in foot-pounds force per slug-degree Rankine) of the gas constant. The default value is for atmospheric air at standard pressure and temperature. This value is used in conjunction with **GAMMA** in determining the gas specific heats at constant pressure and constant volume.

RMACH

(Default Value = 0.5)

```
RMACH = 0.5
```

The **RMACH** keyword value is intended to set an initial or reference flow Mach number. This value is used primarily to set the initial freestream flow variables (density, pressure, temperature and axial velocity) for a given calculation based on a fixed Mach number. The freestream values are used to initialize the flowfield prior to the execution of the time-marching solver in the absence of a restart file. It should be mentioned that the initial data values are assigned based on the assumption that the nondimensional freestream total pressure and total temperature are 1.0 where the nondimensional values are referenced to the dimensional values determined by the **PREF** and **TREF** input variables. This implies that it is advisable to set up all input variables (in particular **PREF** and **TREF**), and boundaray data for **PEXIT** (described in Section 3.7 on boundary data file specifications) such that the imposed inlet and exit flow boundary conditions are compatible with the initial conditions derived from RMACH, based on the assumed global nondimensional total pressure and temperature. For example, suppose that the desired solution for an internal stage compressor rotor has an inlet total pressure of 24 psia, and an exit static pressure of 23.5 psia. For compressor designers, these numbers might commonly be referenced to standard atmospheric pressure (14.7 psia), resulting in nondimensional upstream total and exit static pressures of 1.6326 and 1.5986, respectively. If RMACH is set to 0.5, and the reference pressure is 14.7 psia, then the interior mesh points will be initiated with a static pressure value of 0.84302. It is unlikely that a stable solution will result when the exit static pressure is 1.5986, and the interior static pressure is 0.84302 (reversed flow at the exit boundary will result). A better approach is to specify 24 psia as the reference pressure, such that the nondimensional inlet total and exit static pressures are 1.0, and 0.97917, and the initial nondimensional static pressure at the interior cells is 0.84302. With these values, it is much more likely that a stable solution will result. In addition, the value of **RMACH** is used in conjunction with the value of advance ratio specified by the keyword **ADVR**, when the rotational speed is defined in this manner. In this case, the value of **RMACH** must be the freestream Mach number associated with the advance ratio specified by **ADVR** or an incorrect rotational speed will be calculated. A common error when using the **RMACH** input variable is to assume that the specification of the reference Mach number will set the flow for the case of interest. This is not true, as the boundary condition specifications will ultimately determine the flow conditions for any case.

RPM(*NUM*)

(Default Value = 0.0)

```
RPM(1) = 0.0
RPM(2) = 0.0
RPM(3) = 0.0
RPM(4) = 0.0
RPM(5) = 0.0
```

The **RPM** keyword value determines the rotational speed (in revolutions per minute) of the mesh block number specified by the value *NUM*. The value of **RPM** is, by nature, a dimensional value. Block rotational speeds are, by default, zero, unless either an **RPM** or an **ADVR** keyword are specified otherwise. The user should be aware that if the mesh has not been correctly non-dimensionalized, it is then possible that an incorrect value of rotational speed would be used in the calculation (see the description of the keyword **DIAM**). The user should also be aware that this value is sign specific, and many computational problems traced to geometries which were rotating the wrong way.

TREF

(Default Value = 518.67)

```
TREF = 518.67
```

The **TREF** keyword sets the dimensional value (in degrees Rankine) of the reference total temperature used to nondimensionalize the flowfield. For viscous flows, this value must be accurately specified in order to properly set the nondimensional flow viscosity, (and hence, the Reynolds number). This value is also important for the specification of wall temperature used in the viscous wall boundary condition **SSVI, SS2DVI** (see the description of the boundary data file, Section 3.7). For inviscid flow predictions, this value has no real significance because of the similarity of inviscid flows with Mach number. It is very important to choose an average representative value for this variable, such that the nondimensional total temperature at any point in the flow is near a value of 1.0. An extended discussion on the reason for this choice is given in the description of **RMACH**.

VIS2

(Default Value = 1/2)

```
VIS2 = 0.5
```

The **VIS2** keyword defines the value of the second order added dissipation term used in the fine mesh time-marching solver (see the Final Report [1]). This value is a simple multiplier of the second order dissipation term, and hence, larger values imply more added dissipation. Second order dissipation is used mainly to control the solution in the vicinity of flow discontinuities such as shock waves, but can also be important in any high gradient region. The recommended value is 0.5, but values from 0.0 (no second order dissipation) to 2.0 may be necessary. Any value larger than 2.0 is of questionable use, as the added dissipation will likely dominate the solution.

VIS4

(Default Value = 1/64)

```
VIS4 = 0.015625
```

The **VIS4** keyword defines the value of the fourth order added dissipation term used in the fine mesh time-marching solver (see the Final Report [1]). This value is a simple multiplier of the fourth order dissipation term, and hence, larger values imply more added dissipation. Fourth order dissipation is used mainly to provide a background dissipation to control the odd/even point decoupling associated with centered differencing schemes. The recommended value is 0.015625 (1/64), but values from 0.0 (no fourth order dissipation) to 0.0625 (1/16) may be necessary. Any value larger than 0.0625 is of questionable use, as the added dissipation will likely dominate the solution.

VISCG2

Format:(Default Value = 1/8)

```
VISCG2 = 0.125
```

The **VISCG2** keyword controls the value of the second order added dissipation coefficient for coarse mesh subiterations during the multigrid time-marching solution process. Coarse mesh subiterations utilize a simpler dissipation scheme than the fine mesh time-marching scheme, and therefore, a different damping constant is required. Larger values imply increased added dissipation. The recommended value is **VISCG2** = 0.125, although values from 0.0 (no dissipation) to 1.0 are possible. Values larger than 1.0 are not recommended as the solution would then likely be dominated by the dissipation.

XTRANSS, XTRANPS

(Default Value = 0.0)

```
XTRANSS = 0.0
XTRANPS = 0.0
```

The **XTRANSS, XTRANPS** keywords determine the percentage of axial chord at which transition is forced to occur for the point transition model in the *ADPAC08* body centered mesh turbulence model activated by the keyword **FTURBCHT**. This simplified transition model maintains laminar flow until the percentage of axial chord indicated by **XTRANSS, XTRANPS** is exceeded at which point complete transition is forced. Separate variables are provided for the "suction side" and "pressure side", respectively, of the airfoil in question. The transition model parameters are illustrated in Figure 3.1. Fully turbulent (nontransitional) flows should set **XTRANSS, XTRANPS** to 0.0 (transition occurs at leading edge). Other values must be determined on a case by case basis.

ZETARAT

(Default Value = 0.6)

```
ZETARAT = 0.6
```

The keyword **ZETARAT** controls a parameter used in the eigenvalue scaling operator in the residual smoothing algorithm (see the Final Report [1]). The value of **ZETARAT** represents the exponent the ratio of two coordinate eigenvalues and therefore large values of **ZETARAT** (≥ 0.6) imply increased biased for meshes with large differences in coordinate spacing while small values of **ZETARAT** (≤ 0.5) imply decreased bias for meshes with large differences in coordinate spacing. Normally values between 0.5 and 0.6 are recommended.

3.7 *ADPAC08* Boundary Data File Description

The *ADPAC08* boundary data file contains the user-specifiable parameters which control the application of boundary conditions on the multiple-block mesh during a time-marching solution. These boundary specifications determine the location of solid walls, input/output flow regions, and block-to-block communication paths. Prior to a detailed discussion of the actual boundary condition specifications, several boundary condition application concepts should be explained. It is important to understand how boundary conditions are applied in the *ADPAC08* finite volume solution scheme. Finite volume solution algorithms typically employ the concept of a phantom cell to impose boundary conditions on the external faces of a mesh block. This concept is illustrated graphically for a 2-D mesh representation in Figure 3.3.

A phantom cell is a fictitious neighboring cell which is utilized in the application of boundary conditions on the outer boundaries of a mesh block. Since flow variables cannot be directly specified at a mesh surface in a finite volume solution (the flow variables are calculated and stored at cell centers, where the corners of a cell are described by the 8 surrounding mesh points), the boundary data specified in the phantom cell are utilized to control the flux condition at the cell faces of the outer boundary of the mesh block, and, in turn, satisfy a particular boundary condition. All *ADPAC08* boundary condition specifications provide data values for phantom cells to implement a particular mathematical boundary condition on the mesh. It should be emphasized that the phantom cells are automatically defined within the *ADPAC08* code, and the user need not be concerned about generating fictitious points within the mesh to accommodate the boundary condition application procedure (mesh points need only be generated for the actual flow domain).

Although boundary conditions are imposed at phantom cells in the numerical solution, the boundary specification is still most conveniently *defined* in terms of grid points, not computational cells. An illustration of the boundary specification method for *ADPAC08* is given in Figure 3.4. All boundary conditions are specified in terms of the grid points on either an i=constant, j=constant, or k=constant mesh surface. In practice, these surfaces are typically on the outer boundaries of the mesh block, but it is also possible to impose a boundary on the interior of a mesh block (see the description of the boundary specifications **KILL** and **KIL2D**, below).

The third important aspect of the application of boundary conditions in the *ADPAC08* code involves the order in which boundary conditions are applied. During the execution of the *ADPAC08* code, all boundary conditions are applied to the various mesh blocks in the order in which they are specified

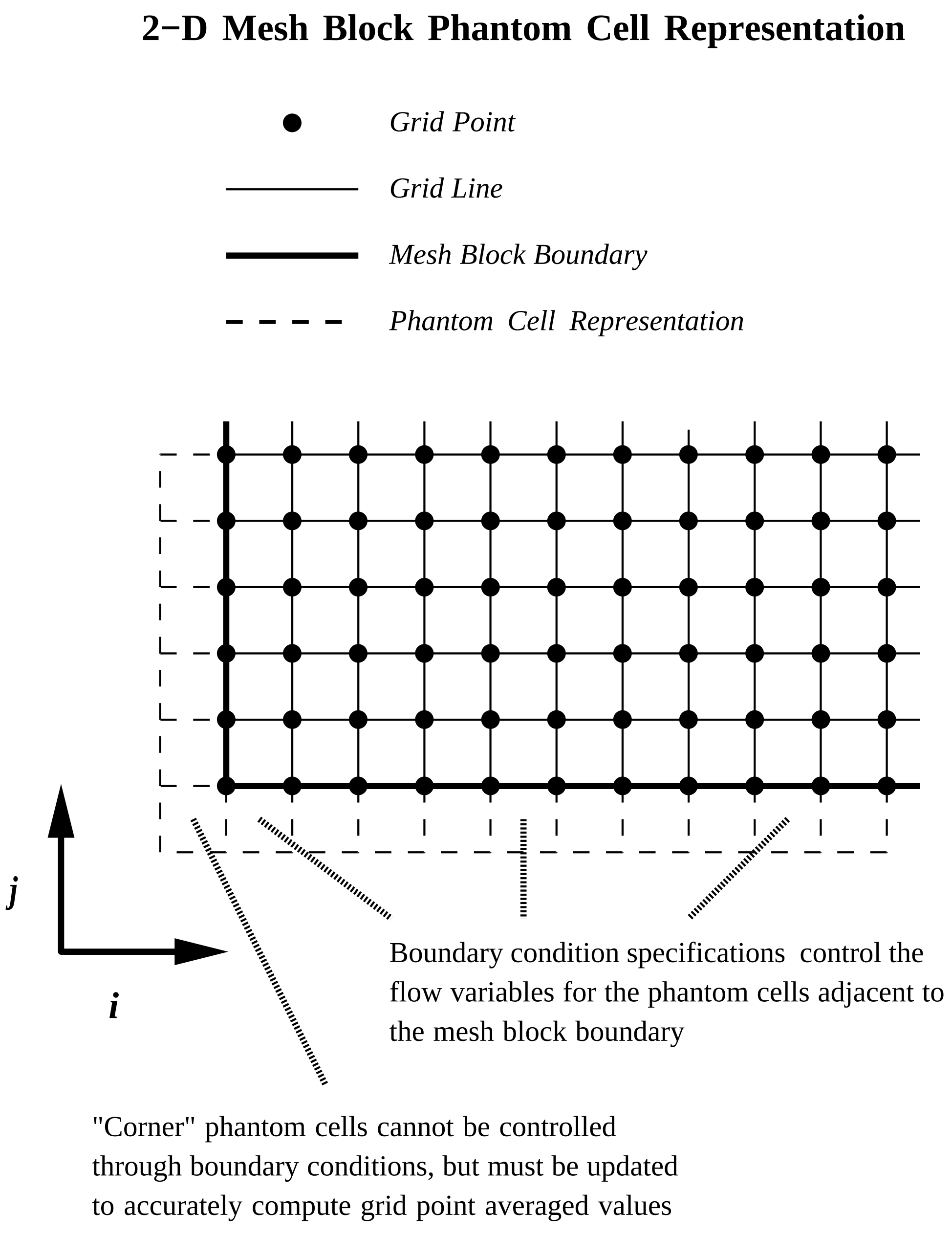
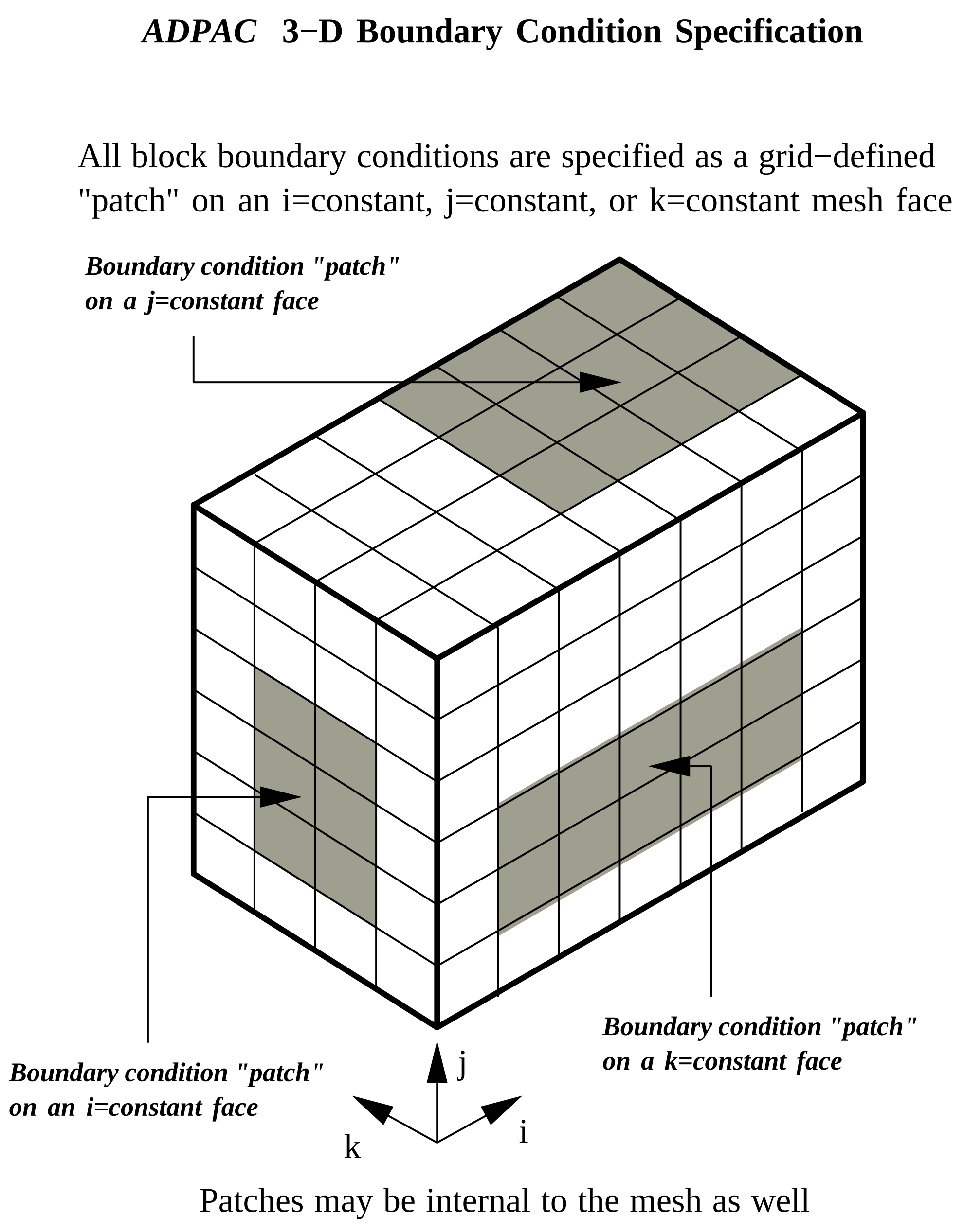

2−D Mesh Block Phantom Cell Representation
● Grid Point
Grid Line
Mesh Block Boundary
Phantom Cell Representation
j
i
Boundary condition specifications control the flow variables for the phantom cells adjacent to the mesh block boundary
"Corner" phantom cells cannot be controlled through boundary conditions, but must be updated to accurately compute grid point averaged values

Figure 3.3: 2-D Mesh Block Phantom Cell Representation



ADPAC 3−D Boundary Condition Specification

All block boundary conditions are specified as a grid−defined "patch" on an i=constant, j=constant, or k=constant mesh face

Boundary condition "patch" on a j=constant face
Boundary condition "patch" on an i=constant face
Boundary condition "patch" on a k=constant face
j
k
i

Patches may be internal to the mesh as well

Figure 3.4: *ADPAC08* 3-D Boundary Condition Specification

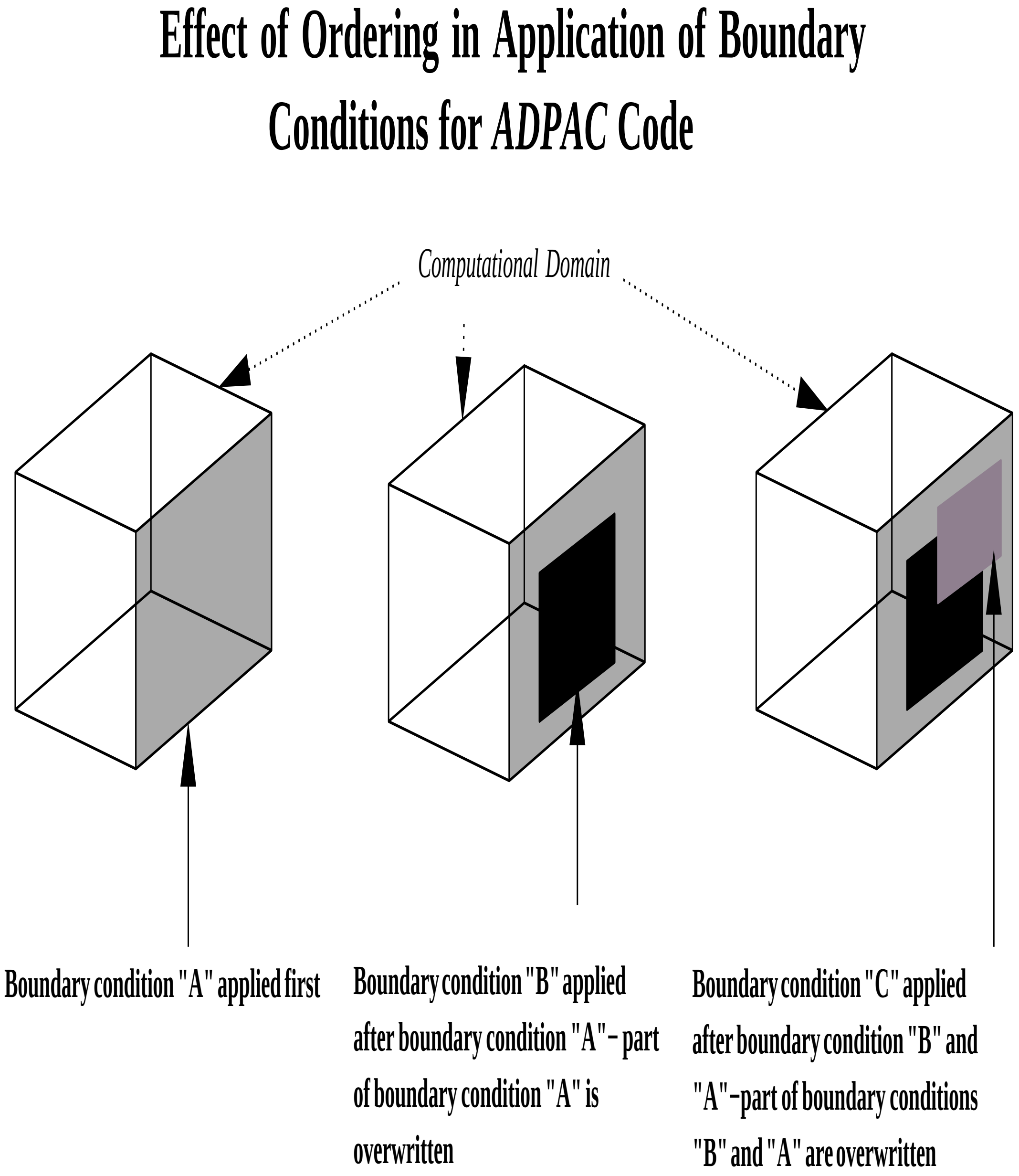

Effect of Ordering in Application of Boundary Conditions for *ADPAC* Code
Computational Domain
Boundary condition "A" applied first
Boundary condition "B" applied after boundary condition "A" - part of boundary condition "A" is overwritten
Boundary condition "C" applied after boundary condition "B" and "A"-part of boundary conditions "B" and "A" are overwritten

Figure 3.5: Effect of Ordering in Application of Boundary Conditions for the *ADPAC08* Code

in the *case*.**boundata** file. As a result, it is possible to overwrite a previously specified boundary patch with a different boundary condition than was originally specified. This concept is illustrated graphically in Figure 3.5. The user must take proper precautions to prohibit accidentally overwriting a desired boundary patch as the *ADPAC08* code cannot distinguish the proper order for the user.

During code execution, the boundary data file is read one line at a time as a character string, and each string is parsed sequentially to determine the specific program action in each case. The boundary data file utilizes a keyword input format, such that any line which does not contain a recognizable keyword is treated as a comment line. Therefore, the user may place any number of comments in the file (so long as the line does not contain a keyword input string in the form described below), and code execution is unaltered. All boundary data file lines are echoed to the standard output, and the program response to each line is listed when a specific action is taken. A line in the boundary data file can also be effectively commented by inserting a # character in the first column. Therefore the lines

```
PATCH    1  1  K  K  P  M  I  J   1  17   1 129   1  17   1 129   1  17
PATCH    2  2  K  K  P  M  I  J   1  17   1 129   1  17   1 129   1  17
```

are acceptable boundary specifications; however, the lines

```
#PATCH    1  1  K  K  P  M  I  J   1  17   1 129   1  17   1 129   1  17
#PATCH    2  2  K  K  P  M  I  J   1  17   1 129   1  17   1 129   1  17
```

would be neglected.

All keyword input lines are given in the format listed in Figure 3.6. The actual specification in the boundary data file may be free format, as long as the individual parameter specifications are given in the correct order and are separated by one or more blank spaces.

All boundary specifications begin with a line containing 19 variables as outlined by the vertical labels in Figure 3.6. A description of the function of each of the variables in the boundary specification line is given in the proper order in the section below:

CASE.BOUNDATA FILE

This file contains case-specific boundary data information. All boundary condition applications are represented by 1 or more lines in this file.

A sample line is given in the highlighted region below:

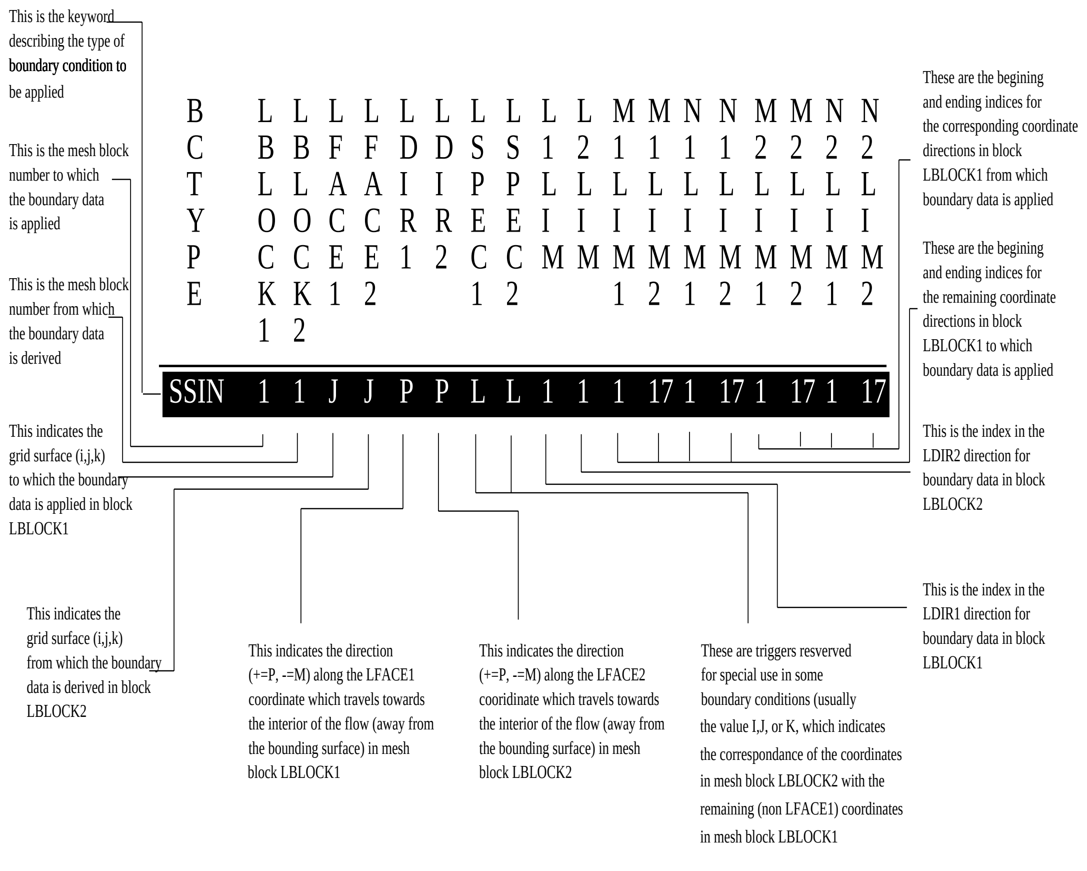

This is the keyword describing the type of boundary condition to be applied
This is the mesh block number to which the boundary data is applied
This is the mesh block number from which the boundary data is derived
This indicates the grid surface (i,j,k) to which the boundary data is applied in block LBLOCK1
This indicates the grid surface (i,j,k) from which the boundary data is derived in block LBLOCK2
BCTYPE LBLOCK1 LBLOCK2 LFACE1 LFACE2 LDIR1 LDIR2 LSPEC1 LSPEC2 L1LIM L2LIM M1LIM1 M1LIM2 N1LIM1 N1LIM2 M2LIM1 M2LIM2 N2LIM1 N2LIM2
SSIN 1 1 J J P P L L 1 1 1 17 1 17 1 17 1 17
This indicates the direction (+=P, -=M) along the LFACE1 coordinate which travels towards the interior of the flow (away from the bounding surface) in mesh block LBLOCK1
This indicates the direction (+=P, -=M) along the LFACE2 cooridinate which travels towards the interior of the flow (away from the bounding surface) in mesh block LBLOCK2
These are triggers resverved for special use in some boundary conditions (usually the value I,J, or K, which indicates the correspondance of the coordinates in mesh block LBLOCK2 with the remaining (non LFACE1) coordinates in mesh block LBLOCK1
These are the begining and ending indices for the corresponding coordinate directions in block LBLOCK1 from which boundary data is applied
These are the begining and ending indices for the remaining coordinate directions in block LBLOCK1 to which boundary data is applied
This is the index in the LDIR2 direction for boundary data in block LBLOCK2
This is the index in the LDIR1 direction for boundary data in block LBLOCK1

Figure 3.6: *ADPAC08* Boundary Data File Specification Format

Description of Boundary Specification Line Variables

BCTYPE The first variable, **BCTYPE**, is a character string defining the type of boundary condition to be applied to a given mesh block. **BCTYPE** must correspond to one of the reserved boundary condition keywords defined later in this section to be a proper boundary specification. If **BCTYPE** is not one of the reserved names, then the boundary specification line is ignored.

LBLOCK1 The variable **LBLOCK1** is an integer defining the grid block number to which the boundary condition implied by **BCTYPE** is applied. Naturally, this implies **LBLOCK1** $\geq$1, and **LBLOCK1** $\leq$ **NBLKS**, where **NBLKS** represents the last mesh block number.

LBLOCK2 The variable **LBLOCK2** is an integer defining the grid block number from which the boundary condition data implied by **BCTYPE** and applied to mesh block **LBLOCK1** is obtained. In some cases, a boundary specification may involve more than one block (patching two blocks together is an example), and the **LBLOCK2** variable is provided for this purpose. The value of the **LBLOCK2** variable is only used in certain routines, but it is a good idea to be consistent in every boundary specification by duplicating the **LBLOCK1** value for the **LBLOCK2** variable if only a single mesh block is involved in a boundary specification (If the boundary specification only involves a single block, then set **LBLOCK2** = **LBLOCK1**).

LFACE1 The variable **LFACE1** is a single character (one of the letters I, J, or K) specifying the grid plane (i=constant, j=constant, or k=constant) to which the boundary condition is applied in block **LBLOCK1**. This specification determines the grid face to which the boundary specification is applied, based on the method by which boundary conditions are implemented in the finite-volume solution scheme (see the discussion and figures above).

LFACE2 The variable **LFACE2** is a single character (one of the letters I, J, or K) specifying the grid plane (i=constant, j=constant, or k=constant) from which the boundary condition data is derived in block **LBLOCK2**. This specification determines the grid face from which the neighboring block boundary data is derived, based on the method by which boundary conditions are implemented in the finite-volume solution scheme (see the discussion and figures above). Naturally, this variable is only useful for boundary specifications involving more than one block. If only one block is involved, simply set **LFACE2** = **LFACE1**.

LDIR1 The variable **LDIR1** is a single character (one of the letters P or M) specifying the direction (P=plus, M=minus) along the **LDIR1** coordinate in **LBLOCK1** which is directed away (towards the interior flow region) from the boundary surface patch. The specification of this variable is normally automatic when the boundary specification is applied to the external surface of a grid block - (**LDIR1** = P when **L1LIM** = 1, and **LDIR1** = M when **L1LIM** = *IMX,JMX*, or *KMX*. (*IMX,JMX,KMX* indicate the maximum indices of the **LBLOCK1** mesh block in the i, j, and k directions, respectively). The intent here is to provide a means of specifying which side of the boundary surface plane the interior computational cells (non-phantom cells) lie on. This specification is made by providing the coordinate direction of the interior computational cells - the phantom cells are then assumed to lie in the opposite direction.

LDIR2 The variable **LDIR2** is a single character (one of the letters P or M) specifying the direction (P=plus, M=minus) along the **LDIR2** coordinate in **LBLOCK2** which is away (towards the interior flow region) from the boundary surface patch. This variable is only used in boundary specifications cases involving more than one mesh block. The specification of this variable is normally automatic when the boundary specification data is obtained from the external surface of a neighboring grid block - (**LDIR2** = P when **L2LIM** = 1, and **LDIR2** = M when **L2LIM** = *IMX,JMX*, or *KMX*. (*IMX,JMX,KMX* indicate the maximum indices of the **LBLOCK2** mesh block in the i, j, and k directions, respectively). The intent here is to provide a means of specifying which side of the boundary surface plane the interior computational cells (non-phantom cells) lie on. This specification is made by providing the coordinate direction of the interior computational cells - the phantom cells are then assumed to lie in the opposite direction. If the boundary specification involves only a single mesh block, then simply set **LDIR2** = **LDIR1**.

LSPEC1 The variable **LSPEC1** is a single character (usually I, J, K, L, or H) which implies some special information about the boundary condition specification. This parameter is boundary condition dependent. The most common application of this variable is in the boundary data file keyword **PATCH**, which provides the cell-to-cell connection for two grid blocks with a mating contiguous surface. For boundary conditions involving more than one mesh block (such as **PATCH**), it is possible that the connection between blocks may involve connections between different grid surfaces, and that the indices in block **LBLOCK2** correspond to a different coordinate in block **LBLOCK1**. In the case of a **PATCH** boundary condition, the **LSPEC1** variable determines the grid coordinate direction in the **LBLOCK1** mesh block which corresponds with the first remaining grid coordinate in mesh block **LBLOCK2**. (The extent of the first remaining coordinate in mesh block **LBLOCK2** is determined by the values of **M2LIM1** and **M2LIM2**)

LSPEC2 The variable **LSPEC2** is a single character (usually I, J, K, L, or H) which implies some special information about the boundary condition specification. This parameter is usually boundary condition dependent. The most common application of this variable is in the boundary data file keyword **PATCH**, which provides the cell-to-cell connection for two grid blocks with a mating contiguous surface. For boundary conditions involving more than one mesh block (such as **PATCH**), it is possible that the connection between blocks may involve connections between different grid surfaces, and that the indices in block **LBLOCK2** correspond to a different coordinate in block **LBLOCK1**. In the case of a **PATCH** boundary condition, the **LSPEC2** variable determines the grid coordinate direction in the **LBLOCK1** mesh block which corresponds with the second remaining grid coordinate in mesh block **LBLOCK2**. (The extent of the second remaining coordinate in mesh block **LBLOCK2** is determined by the values of **N2LIM1** and **N2LIM2**)

L1LIM The variable **L1LIM** is an integer specifying the index of the grid in the **LFACE1** direction to which the boundary condition should be applied in block **LBLOCK1**. This value determines the actual mesh index of the i=constant, j=constant, or k=constant mesh face (determined by **LFACE1**) to which the boundary condition is applied in mesh block **LBLOCK1**.

L2LIM The variable **L2LIM** is an integer specifying the index of the grid in the **LFACE2** direction from which the boundary condition data is derived in block **LBLOCK2**. This value determines the actual mesh index of the i=constant, j=constant, or k=constant mesh face (determined by **LFACE2**) from which the boundary condition data is derived in mesh block **LBLOCK2**.

M1LIM1 The variable **M1LIM1** is an integer representing the initial index of the first remaining grid coordinate direction to which the boundary condition is applied in block **LBLOCK1**. Since the boundary specification applies to either an i=constant, j=constant, or k=constant surface, the variables **M1LIM1, M1LIM2, N1LIM1** and **N1LIM2** determine the extent of the patch in the remaining coordinate directions. The remaining coordinate directions for block **LBLOCK1** are specified in the natural order. (For example, if **LFACE1**=I, then the variables **M1LIM1, M1LIM2** refer to the extent in the j direction and the variables **N1LIM1, N1LIM2** refer to the extent in the k direction. If **LFACE1**=J, then the variables **M1LIM1, M1LIM2** refer to the extent in the i direction and the variables **N1LIM1, N1LIM2** refer to the extent in the k direction. If **LFACE1**=K, then the variables **M1LIM1, M1LIM2** refer to the extent in the i direction and the variables **N1LIM1, N1LIM2** refer to the extent in the j direction.) The indices specified in **M1LIM1** and **M1LIM2** must be given in increasing order. The indices specified in **N1LIM1** and **N1LIM2** must also be given in increasing order.

M1LIM2 The variable **M1LIM2** is an integer representing the final index of the first remaining grid coordinate direction to which the boundary condition is applied in block **LBLOCK1**. Since the boundary specification applies to either an i=constant, j=constant, or k=constant surface, the variables **M1LIM1, M1LIM2, N1LIM1** and **N1LIM2** determine the extent of the patch in the remaining coordinate directions. The remaining coordinate directions for block **LBLOCK1** are specified in the natural order. (For example, if **LFACE1**=I, then the variables **M1LIM1, M1LIM2** refer to the extent in the j direction and the variables **N1LIM1, N1LIM2** refer to the extent in the k direction. If **LFACE1**=J, then the variables **M1LIM1, M1LIM2** refer to the extent in the i direction and the variables **N1LIM1, N1LIM2** refer to the extent in the k direction. If **LFACE1**=K, then the variables **M1LIM1, M1LIM2** refer to the extent in the i direction and the variables **N1LIM1, N1LIM2** refer to the extent in the j direction.) The indices specified in **M1LIM1** and **M1LIM2** must be given in increasing order. The indices specified in **N1LIM1** and **N1LIM2** must also be given in increasing order.

N1LIM1 The variable **N1LIM1** is an integer representing the initial index of the second remaining grid coordinate direction to which the boundary condition is applied in block **LBLOCK1**. Since the boundary specification applies to either an i=constant, j=constant, or k=constant surface, the variables **M1LIM1, M1LIM2, N1LIM1** and **N1LIM2** determine the extent of the patch in the remaining coordinate directions. The remaining coordinate directions for block **LBLOCK1** are specified in the natural order. (For example, if **LFACE1**=I, then the variables **M1LIM1, M1LIM2** refer to the extent in the j direction and the variables **N1LIM1, N1LIM2** refer to the extent in the k direction. If **LFACE1**=J, then the variables **M1LIM1, M1LIM2** refer to the extent in the i direction and the variables **N1LIM1, N1LIM2** refer to the extent in the k direction. If **LFACE1**=K, then the variables **M1LIM1, M1LIM2** refer to the extent in the i direction and the variables **N1LIM1, N1LIM2** refer to the extent in the j direction.) The indices specified in **M1LIM1** and **M1LIM2** must be given in increasing order. The indices specified in **N1LIM1** and **N1LIM2** must also be given in increasing order. For boundaries on 2-D mesh blocks, this must always be 1.

N1LIM2 The variable **N1LIM2** is an integer representing the final index of the second remaining grid coordinate direction to which the boundary condition is applied in block **LBLOCK1**. Since the boundary specification applies to either an i=constant, j=constant, or k=constant surface, the variables **M1LIM1, M1LIM2, N1LIM1** and **N1LIM2** determine the extent of the patch in the remaining coordinate directions. The remaining coordinate directions for block **LBLOCK1** are specified in the natural order. (For example, if **LFACE1**=I, then the variables **M1LIM1, M1LIM2** refer to the extent in the j direction and the variables **N1LIM1, N1LIM2** refer to the extent in the k direction. If **LFACE1**=J, then the variables **M1LIM1, M1LIM2** refer to the extent in the i direction and the variables **N1LIM1, N1LIM2** refer to the extent in the k

direction. If **LFACE1**=K, then the variables **M1LIM1, M1LIM2** refer to the extent in the i direction and the variables **N1LIM1, N1LIM2** refer to the extent in the j direction.) The indices specified in **M1LIM1** and **M1LIM2** must be given in increasing order. The indices specified in **N1LIM1** and **N1LIM2** must also be given in increasing order. For boundaries on 2-D mesh blocks, this must always be 2.

M2LIM1 The variable **M2LIM1** is an integer representing the initial index of the grid coordinate direction in block **LBLOCK2** corresponding to the first remaining coordinate in block **LBLOCK1**. For boundary conditions involving more than one mesh block, it is possible that the connection between blocks may involve connections between different grid surfaces, and that the indices in block **LBLOCK2** correspond to a different coordinate in block **LBLOCK1**. The variables **M2LIM1, M2LIM2** control the indices in the **LSPEC1** direction in block **LBLOCK2** which correspond to the indices determined by **M1LIM1, M1LIM2** in block **LBLOCK1**. The user should note that it is possible for **M2LIM1** > **M2LIM2** and **N2LIM1** > **N2LIM2** but it is not possible for **M1LIM1** > **M1LIM2** and **N1LIM1** > **N1LIM2**. If only a single mesh block is involved in the boundary specification, set **M2LIM1** = **M1LIM1**.

M2LIM2 The variable **M2LIM2** is an integer representing the final index of the grid coordinate direction in block **LBLOCK2** corresponding to the first remaining coordinate in block **LBLOCK1**. For boundary conditions involving more than one mesh block, it is possible that the connection between blocks may involve connections between different grid surfaces, and that the indices in block **LBLOCK2** correspond to a different coordinate in block **LBLOCK1**. The variables **M2LIM1, M2LIM2** control the indices in the **LSPEC1** direction in block **LBLOCK2** which correspond to the indices determined by **M1LIM1, M1LIM2** in block **LBLOCK1**. The user should note that it is possible for **M2LIM1** > **M2LIM2** and **N2LIM1** > **N2LIM2** but it is not possible for **M1LIM1** > **M1LIM2** and **N1LIM1** > **N1LIM2**. If only a single mesh block is involved in the boundary specification, set **M2LIM2** = **M1LIM2**.

N2LIM1 The variable **N2LIM1** is an integer representing the initial index of the grid coordinate direction in block **LBLOCK2** corresponding to the second remaining coordinate in block **LBLOCK1**. For boundary conditions involving more than one mesh block, it is possible that the connection between blocks may involve connections between different grid surfaces, and that the indices in block **LBLOCK2** correspond to a different coordinate in block **LBLOCK1**. The variables **N2LIM1, N2LIM2** control the indices in the **LSPEC2** direction in block **LBLOCK2** which correspond to the indices determined by **N1LIM1, N1LIM2** in block **LBLOCK1**. The user should note that it is possible for **M2LIM1** > **M2LIM2** and **N2LIM1** > **N2LIM2** but it is not possible for **M1LIM1** > **M1LIM2** and **N1LIM1** > **N1LIM2**. If only a single mesh block is involved in the boundary specification, set **N2LIM1** = **N1LIM1**. For boundary data on 2-D mesh blocks, this must always be 1.

N2LIM2 The variable **N2LIM2** is an integer representing the final index of the grid coordinate direction in block **LBLOCK2** corresponding to the second remaining coordinate in block **LBLOCK1**. For boundary conditions involving more than one mesh block, it is possible that the connection between blocks may involve connections between different grid surfaces, and that the indices in block **LBLOCK2** correspond to a different coordinate in block **LBLOCK1**. The variables **N2LIM1, N2LIM2** control the indices in the **LSPEC2** direction in block **LBLOCK2** which correspond to the indices determined by **N1LIM1, N1LIM2** in block **LBLOCK1**. The user should note that it is possible for **M2LIM1** > **M2LIM2** and **N2LIM1** > **N2LIM2** but it is not possible for **M1LIM1** > **M1LIM2** and **N1LIM1**

> **N1LIM2**. If only a single mesh block is involved in the boundary specification, set **N2LIM2** = **N1LIM2**. For boundary data on 2-D mesh blocks, this must always be 2.

Some boundary condition specifications require additional data beyond that incorporated in the boundary specification line. In these cases, described in detail for the specific boundary types later in this Section, the additional data is included immediately after the boundary specification line.

A sample *ADPAC08* boundary data file containing several keywords is listed below.

Sample *ADPAC-AOACR* Boundary Data File

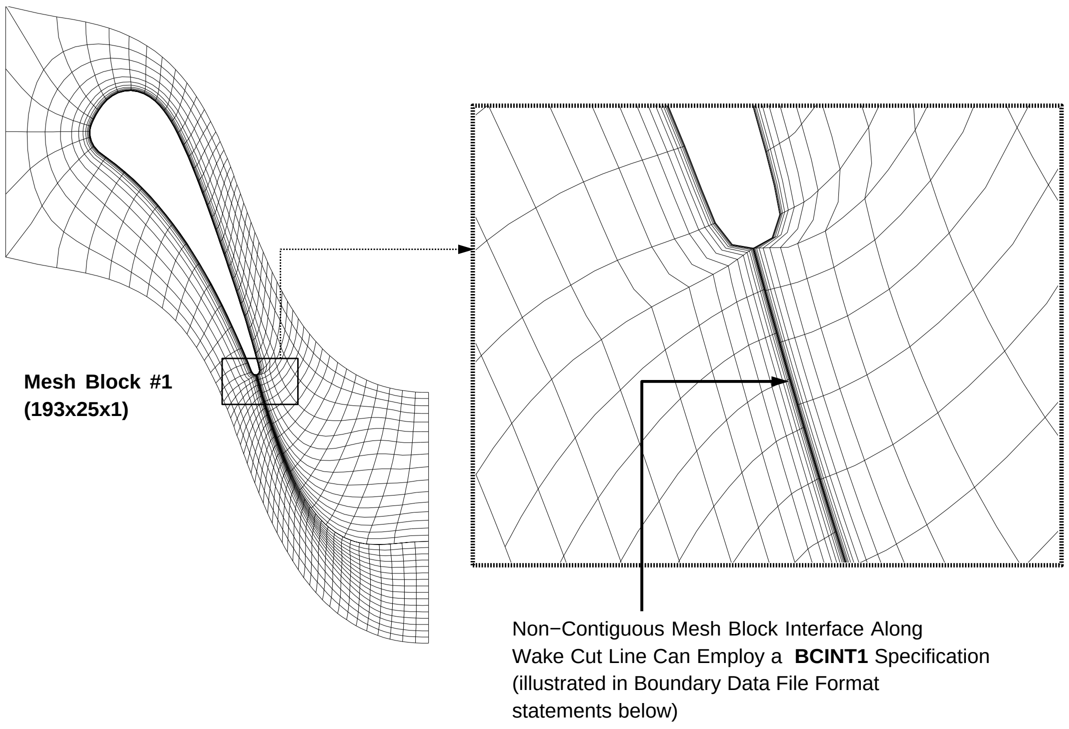
```
-----------------------------------------------------
ADPAC Boundata File Generated by SETUP-Version 1.0
Apr 15 1992, at 17:36:38
-----------------------------------------------------
 This file contains block boundary condition data information for the
 ADPAC-AOACR multiple grid block Euler-Navier-Stokes code. All boundary
 specifications begin with a line containing the variables:

The routine selected is for a two-block 3-D H-grid
about a ducted propfan as shown below:

                    I-J Plane

J=JL ------------------------------------------------
     |Block #2           COWL|<--MIXING PLANE       |
     |           I=IGCLE   | |    I=IGCTE           |
     |   (COWL L.E.)_______V_|______ (COWL T.E.)    |
     |-------------<________________>---------------|
     |                ____   |   ____               |
     |         J=JTIP/    |  |  /   |<--- BLADE     |
     |               /    /  | /    /               |
     |              |    /   | |    /               |
     |Block #1    ---------------------             |
     |           /  I=ILE  I=ITE        |          |
J=1  -----------/  LEADING TRAILING      -----------
    I=1    I=ISLE   EDGE    EDGE      I=ISTE      I=IL
           SPINNER                    SPINNER
         LEADING EDGE              TRAILING EDGE


                    I-K Plane

K=KL -------------        -----     ------------
     |            |______|  | |_____|          |
     |                A     |    A             |
     |                |     |    |             |
     |    BLADE SURFACE     |  BLADE SURFACE   |
     |                |     |    |             |
     | (Odd # blocks  |     |    |             |
     |  only)         |     |    |             |
     |                |     |<-------- MIXING  |
     |                |     |    |     PLANE   |
     |             ___V__   |  __V__           |
K=1  -------------|      |----|     |-----------
    I=1            I=ILE   I=ITE                  I=IL

BLOCKDATA FOLLOWS:              LABELS

B       L  L  L  L  L  L  L  L   L   L   M   M   N   N   M   M   N   N     C
C       B  B  F  F  D  D  S  S   1   2   1   1   1   1   2   2   2   2     O
T       L  L  A  A  I  I  P  P   L   L   L   L   L   L   L   L   L   L     M
Y       O  O  C  C  R  R  E  E   I   I   I   I   I   I   I   I   I   I     M
P       C  C  E  E  1  2  C  C   M   M   M   M   M   M   M   M   M   M     E
E       K  K  1  2        1  2           1   2   1   2   1   2   1   2     N
        1  2                                                               T
------ -- -- -- -- -- -- -- -- --- --- --- --- --- --- --- --- --- --- ---------
PATCH   1  1  K  K  P  M  I  J   1  17   1 129   1  17   1 129   1  17 %K=1 Per
PATCH   2  2  K  K  P  M  I  J   1  17   1 129   1  17   1 129   1  17 %K=1 Per
PATCH   3  3  K  K  P  M  I  J   1  17   1  97   1  17   1  97   1  17 %K=1 Per
PATCH   4  4  K  K  P  M  I  J   1  17   1  97   1  17   1  97   1  17 %K=1 Per
PATCH   1  1  K  K  M  P  I  J  17   1   1 129   1  17   1 129   1  17 %K=KL Pe
PATCH   2  2  K  K  M  P  I  J  17   1   1 129   1  17   1 129   1  17 %K=KL Pe
PATCH   3  3  K  K  M  P  I  J  17   1   1  97   1  17   1  97   1  17 %K=KL Pe
PATCH   4  4  K  K  M  P  I  J  17   1   1  97   1  17   1  97   1  17 %K=KL Pe
SSIN    1  1  J  J  P  P  S  S   1   1   1 129   1  17   1 129   1  17 %Hub Inv
SSIN    3  3  J  J  P  P  S  S   1   1   1  97   1  17   1  97   1  17 %Hub Inv
SSIN    1  1  K  K  P  P  S  S   1   1  81 113   1  17  81 113   1  17 %K=1  Bl
SSIN    1  1  K  K  M  M  S  S  17  17  81 113   1  17  81 113   1  17 %K=KL Bl
SSIN    3  3  K  K  P  P  S  S   1   1  17  49   1  17  17  49   1  17 %K=1  Bl
SSIN    3  3  K  K  M  M  S  S  17  17  17  49   1  17  17  49   1  17 %K=KL Bl
#INLETA  1  1  I  I  P  P  S  S   1   1   1  17   1  17   1  17   1  17 %Inlet
  PT        TT        ALPHA
  1.000000  1.000000  0.000000
#INLETA  2  2  I  I  P  P  S  S   1   1   1  17   1  17   1  17   1  17 %Inlet
  PT        TT        ALPHA
  1.000000  1.000000  0.000000
EXITP   3  3  I  I  M  M  H  H  97  97   1  17   1  17   1  17   1  17 %Inlet
NBOUN     IBOUN     JBOUN
    000004    000097    000001
EXITT   4  4  I  I  M  M  H  H  97  97   1  17   1  17   1  17   1  17 %Inlet
  PEXIT
  0.3609139
FREE    2  2  J  J  M  M  S  S  17  17   1 129   1  17   1 129   1  17 %Free fl
  PT        TT        MACH(INF)  ALPHA
  1.000000  1.000000  1.300000  0.000000
FREE    4  4  J  J  M  M  S  S  17  17   1  97   1  17   1  97   1  17 %Free fl
  PT        TT        MACH(INF)  ALPHA
  1.000000  1.000000  1.300000  0.000000
PATCH   1  2  J  J  M  P  I  J  17   1   1 129   1  17   1 129   1  17 %Ptch 1-
PATCH   2  1  J  J  P  M  I  J   1  17   1 129   1  17   1 129   1  17 %Ptch 1-
PATCH   3  4  J  J  M  P  I  J  17   1   1  97   1  17   1  97   1  17 %Ptch 1-
PATCH   4  3  J  J  P  M  I  J   1  17   1  97   1  17   1  97   1  17 %Ptch 1-
SSIN    1  1  J  J  M  M  S  S  17  17  33 129   1  17  33 129   1  17 %COWL 1
SSIN    2  2  J  J  P  P  S  S   1   1  33 129   1  17  33 129   1  17 %COWL 2
SSIN    3  3  J  J  M  M  S  S  17  17   1  81   1  17   1  81   1  17 %COWL 1
SSIN    4  4  J  J  P  P  S  S   1   1   1  81   1  17   1  81   1  17 %COWL 2
MBCAVG  1  3  I  I  M  P  J  K 129   1   1  17   1  17   1  17   1  17 %Mixing
```

```
NSEGS
1
BLK   LFACE2 LDIR2 L2LIM M2LIM1 M2LIM2 N2LIM1 N2LIM2
3     I      P     1     1      17     1      17
MBCAVG  2  4  I  I  M  P  J  K 129   1   1  17   1  17   1  17   1  17 %Mixing
NSEGS
1
BLK   LFACE2 LDIR2 L2LIM M2LIM1 M2LIM2 N2LIM1 N2LIM2
4     I      P     1     1      17     1      17
MBCAVG  3  1  I  I  P  M  J  K   1 129   1  17   1  17   1  17   1  17 %Mixing
NSEGS
1
BLK   LFACE2 LDIR2 L2LIM M2LIM1 M2LIM2 N2LIM1 N2LIM2
1     I      M   129     1      17     1      17
MBCAVG  4  2  I  I  P  M  J  K   1 129   1  17   1  17   1  17   1  17 %Mixing
NSEGS
1
BLK   LFACE2 LDIR2 L2LIM M2LIM1 M2LIM2 N2LIM1 N2LIM2
2     I      M   129     1      17     1      17
ENDDATA
```

A list and description of all valid boundary data keywords and any additional data required for the given boundary condition is now presented in the pages which follow. A quick reference to the boundary data file keywords is provided in Appendix C.

BCINT1

BCINT1 Type Non-Contiguous Mesh Block Interface Patching Scheme


Mesh Block #1
(193x25x1)
Non−Contiguous Mesh Block Interface Along Wake Cut Line Can Employ a **BCINT1** Specification (illustrated in Boundary Data File Format statements below)

Application

The **BCINT1** specification is used in any application involving neighboring mesh blocks with a non-contiguous interface in one direction. The interface must be contiguous in the other direction. **BCINT1** patches one block to one other block by interpolation along the non-contiguous index.

The example graphic above illustrates a two-dimensional mesh system used to predict the flow through a turbine vane passage. The C-type mesh utilizes a noncontiguous wake cut line as shown in the trailing edge detail. The **BCINT1** specification is applied along either side of the wake cut line to permit communication of flow variables across the noncontiguous mesh interface. Here, the interpolation direction is i, and part of the block is patched to itself. Note that the i index increases in different directions at the wake cut line. **BCINT1** can handle interpolation along any index, regardless of the orientation of the mating surface.

Boundary Data File Format

The boundary data file specifications for the mesh interface indicated in the illustrative graphic for the **BCINT1** boundary condition are given below:

```
BCINT1  1  1  J  J  P  P  L  L   1   1   1  33   1   2 193 177   1   2
  IDIRNT1 IDIRNT2
     I       I
BCINT1  1  1  J  J  P  P  L  L   1   1 177 193   1   2  33   1   1   2
  IDIRNT1 IDIRNT2
     I       I
```

Note that a complete **BCINT1** specification generally requires two **BCINT1** statement lines in the boundary data file. In the example above, the first specification provides the interblock communication for one side of the C-grid wake cut, while the second specification provides the communication for the other side of the C-grid wake cut. It is a common error to underspecify a **BCINT1** boundary by only providing a single line per interface.

Description

The **BCINT1** boundary statement provides a means for block to block communication for cases involving neighboring mesh boundaries which share a common surface, but are non-contiguous in one grid index. **BCINT1** can be applied to either stationary or rotating block interfaces, but the results are physically correct only if both blocks are rotating at the same speed.

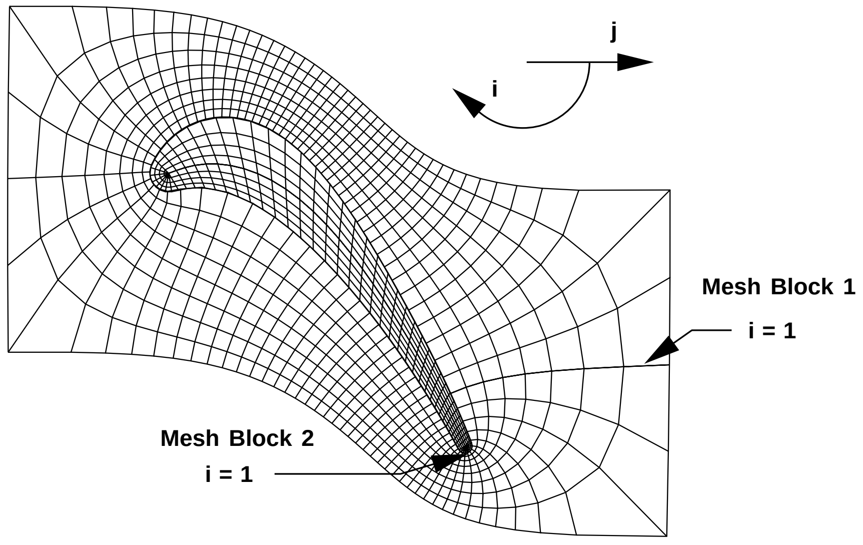
(The **BCPRR** specification should be used for cases with relatively rotating blocks.) A proper **BCINT1** boundary is specified much like a **PATCH** boundary. The **LFACE1** and **LFACE2** determine which faces are mated together. **BCINT1** also requires the specification of additional information. The first line after the **BCINT1** specification may contain any information; however, for consistency, it is recommended that the labels **INTDIR1** and **INTDIR2** be used. The non-contiguous index in each block is the interpolation direction, which is specified as **INTDIR1** or **INTDIR2**. The blocks are assumed to be contiguous in the remaining index. The **M2LIM** or **N2LIM** variables are specified much as they would be for a **PATCH** specification. The exception is that the number of points spanned by the limits in the direction of interpolation need not be equal.

The search routine which determines the interpolation stencil assumes that the mating grid lines are piecewise linear approximations to the same curve in the interpolation direction. A global search is performed for the proper mating cell of the first index. The closest cell to the point of interest is taken as the mating cell. A localized search is performed for the mating cells of the remaining points. The local search starts at the mating cell of the preceding point and searches along the mating boundary until the mating cell containing the new point is found. In the event that the mating cell is not found before the upper limit is reached in the mating block, the search continues from the lower limit in the mating block. This implies two things: the physical domain of the interpolation must be the same in the two blocks, and the domain is assumed to be periodic if the search routine goes past an endpoint.

Restrictions/Limitations

The **BCINT1** boundary specification is restricted to mesh interfaces which lie on a common surface (no significant overlap).

Generally, endpoints of the interpolated region in the two blocks should be coincident. There is at least one exception to this rule based on the above description of the search routine. In the case of concentric O-grids, the endpoints of the two blocks may be misaligned as shown in the figure below. The interpolation routine will find the appropriate stencil for each point because the grids are periodic.



The **BCINT1** condition reduces to a **PATCH** condition if the mating blocks are actually contiguous. However, due to the linear interpolation used in **BCINT1**, the scheme does not maintain either global or local conservation of flow variables across a non-contiguous mesh interface.

The **BCINT1** condition also performs the same function as the **TRAF** condition, but with fewer restrictions. The **TRAF** condition employs a cubic spline for interpolation, rather than the linear procedure used by **BCINT1**. The **BCINT1** condition is the only non-contiguous patching routine which may be utilized across multiple processors in a parallel computing environment.

Common Errors

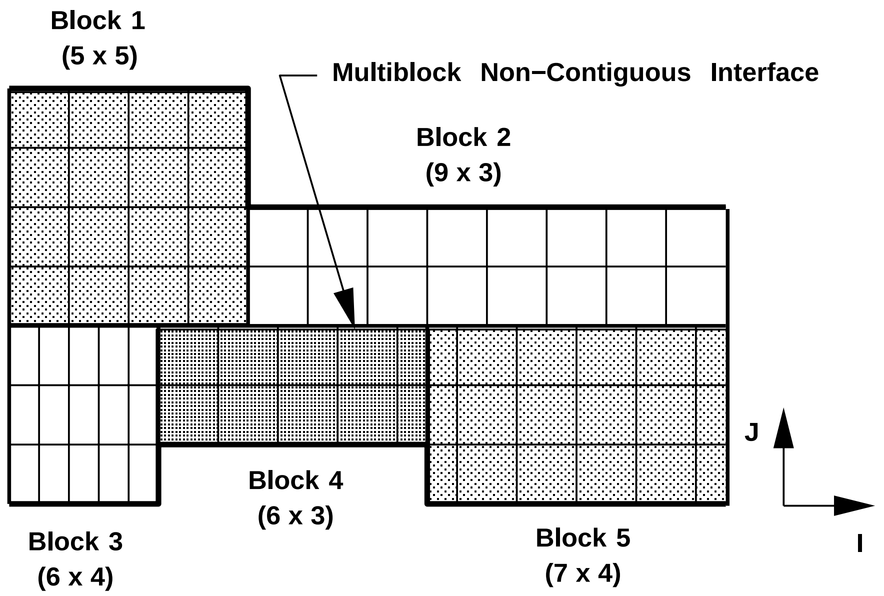
- Failure to provide 2 **BCINT1** statements for each interface.
- Incorrectly specified or misaligned extents of boundary regions (values of **M1LIM1, M1LIM2, N1LIM1, N1LIM2, M2LIM1, M2LIM2,**

N2LIM1, N2LIM2 do not correctly define the interface extents on blocks **LBLOCK1** and **LBLOCK2**).

- Incorrectly specified index for the interpolation direction for **LBLOCK1** or **LBLOCK2**.
- Attempt to use **BCINT1** for a boundary which has 2 misaligned coordinates.
- Attempt to use **BCINT1** for boundaries which are not monotonic along the interpolated index.

BCINTM

BCINTM Type Non-Contiguous Mesh Block Interface Patching Scheme



Non−contiguous mesh block interface involving multiple blocks requires a **BCINTM** specification (illustrated in boundary data file format statements below

Application

The **BCINTM** specification is used in any application involving neighboring mesh blocks with a non-contiguous mesh interface in one coordinate direction. The interface must be contiguous in the remaining coordinate direction. **BCINTM** provides a mechanism whereby noncontiguous boundaries involving groups of blocks may be coupled to other groups of blocks by interpolation along the non-contiguous index. **BCINTM** is a multi-block version of **BCINT1**.

The example graphic above illustrates a two-dimensional mesh system used to predict the flow through a stepped duct passage. The grid was constructed with a non-contiguous interface between the various blocks on the top and bottom of the duct. The **BCINTM** specification is applied along either side of the interface to permit communication of flow variables along the interface. Here, the interpolation direction is the i coordinate direction.

Boundary Data File Format

The boundary data file specifications for the mesh interface indicated in the illustrative graphic for the **BCINTM** boundary condition are given below:

```
BCINTM     1   3  J  J  P  M  I  K   1   4   1   5   1   1   1   5   1   1
 INTDIR1 INTDIR2 - DIRECTION OF INTERPOLATION
       I       I
   ISHFTDR    DSHIFT
         2     0.0
 NBLINT2 - NUMBER OF LBLOCK2 BLOCKS
       3
 NBLDAT  LFACE2 LDIR2 L2LIM M2LIM1 M2LIM2 N2LIM1 N2LIM2
    3        J     M     4      1      5      1      1
    4        J     M     3      1      6      1      1
    5        J     M     4      1      7      1      1
 NBLINT1 - NUMBER OF LBLOCK1 BLOCKS
       2
 LBLK1RR LFACE1 LDIR1 L1LIM M1LIM1 M1LIM2 N1LIM1 N1LIM2
    1        J     P     1      1      5      1      1
    2        J     P     1      1      9      1      1
BCINTM     3   1  J  J  M  P  I  K   4   1   1   5   1   1   1   5   1   1
 INTDIR1 INTDIR2 - DIRECTION OF INTERPOLATION
       I       I
   ISHFTDR    DSHIFT
         2     0.0
 NBLINT2 - NUMBER OF LBLOCK2 BLOCKS
```

```
      2
NBLDAT  LFACE2 LDIR2 L2LIM M2LIM1 M2LIM2 N2LIM1 N2LIM2
   1         J     P     1      1      5      1      1
   2         J     P     1      1      9      1      1
NBLINT1 - NUMBER OF LBLOCK1 BLOCKS
      3
LBLK1RR LFACE1 LDIR1 L1LIM M1LIM1 M1LIM2 N1LIM1 N1LIM2
   3         J     M     4      1      5      1      1
   4         J     M     3      1      6      1      1
   5         J     M     4      1      7      1      1
```

Note that a complete **BCINTM** specification generally requires two **BCINTM** statement lines in the boundary data file. In the example above, the first specification provides the interblock communication for the upper blocks along the interface, while the second specification provides the communication for the lower blocks along the interface. It is a common error to underspecify a **BCINTM** boundary by only providing a single line per interface.

Description

The **BCINTM** boundary statement provides a means for block to block communication for cases involving neighboring mesh boundaries which share a common surface, but are non-contiguous in one grid index. A proper **BCINTM** boundary is specified much like a **BCINT1** boundary, except that all of the blocks involved with a particular interface are specified in a table on both sides of the interface. A large amount of additional data is required for each **BCINTM** specification. The sample application and specifications given above are designed to demonstrate the overall structure of this boundary condition. In the sample application, a noncontiguous mesh block interface lies between blocks 1,2 and blocks 3,4,5. A single pair of **BCINTM** specifications is all that is required to completely couple the mesh blocks along this interface, in spite of the fact that 5 mesh blocks are involved in the overall boundary definition. The key to this compact specification is that each **BCINTM** specification includes tables of data which specify which blocks lie along the receiving side of the interface (where the boundary data is being applied) and which blocks lie along the sending side of the interface (where the boundary data is derived). A description of the various additional specifications required for a complete **BCINTM** specification are given below.

Immediately following the **BCINTM** boundary specification line is a series of multi-line segments which define the details of the boundary coupling. The first segment consists of 4 lines, and describes some general characteristics of the interpolation along the noncontiguous boundary. The second segment is the table describing the "sending" blocks from which the boundary data is extracted. The third segment is a table describing the "receiving" blocks where the boundary data is eventually interpolated and applied. The second line in a **BCINTM** specification is a comment line, normally labeling the variables **INTDIR1** and **INTDIR2**. The third line defines the variables **INTDIR1** and **INTDIR2** as either I, J, or K, depending on the direction of interpolation for the receiving and sending blocks, respectively. One mesh restriction to note is that **BCINTM** allows only one interpolation direction for each side of the interface. The fourth line is a comment line normally labeling the variables **ISHFTDR** and **DSHIFT**. The fifth line defines the values for the variables **ISHFTDR** and **DSHIFT**. These variables provide a mechanism for shifting the boundary in on of the three coordinate directions (x, y, z for Cartesian flows, or z, r, θ for cylindrical flows). **BCINTM** expects that the two sides of the interface lie on a common physical surface, but the grid itself may not have both sides of the interface in the same physical location. The most common use for this feature is a noncontiguous periodic boundary for a single passage turbomachinery blade row solution. The **ISHFTDR** and **DSHIFT** variables are provided to allow the user to temporarily shift the physical location of the "sending" blocks to the "receiving" blocks physical location. For the case of a noncontiguous periodic boundary in a turbine blade row solution, for example, **ISHFTDR** would be 3 (shift in the θ direction) and the amount of the shift defined by **DSHIFT** would be the circumferential spacing of the blade rows in radians. The sixth line in the specification is a comment normally labeling the variable **NBLINT2**, and the seventh line specifies the number of blocks associated with the **LBLOCK2** side of the interface (the "sending" blocks. The eigth line is again a comment normally labeling the variables **NBLDAT**, **LFACE2**, **LDIR2**, **L2LIM**, **M2LIM1**, **M2LIM2**, **N2LIM1**, and

N2LIM2. The next **NBLINT2** lines define the table containing the limits, directions and faces for each of the **LBLOCK2** blocks. For each block in this table, **LFACE2** defines the coordinate face upon which the interface lies I, J, or K), and **LDIR2** defines the direction (P for plus, or M for minus) along the **LFACE2** coordinate which travels away from the bounding surface (see Section 3.7 for more details). **L2LIM** defines the value of the **LFACE2** coordinate upon which the surface is located, and **M2LIM1**, **M2LIM2**, and **N2LIM1**, and **N2LIM2** define the extent of the remaining coordinates for each of the **NBLINT2** blocks in their "natural" order (again see Section 3.7 for more details). Following the table for the **LBLOCK2** side of the interface, there is a commment line normally labeling the variable **NBLINT1**, followed by a line specifying the number of blocks on the **LBLOCK1** side of the interface. Next a comment line labeling the variables **L1LIM**, **M1LIM1**, **M1LIM2**, **N1LIM1**, and **N1LIM2** is given. Finally, a table consisting of **NBLINT1** lines defining the **LBLOCK1** side ("receiving" blocks) information similar to the **LBLOCK2** ("sending" blocks) table is specified.

BCINTM creates a single interpolation stencil from all of the blocks in the **LBLOCK2** table. This stencil must be monotonic in the **INTDIR2** direction. Thus, the blocks in the **LBLOCK2** must be specified in the order they occur physically, and the limits must be specified so that they form a continous line. The block numbers and extents identified in the first line of the **BCINTM** specification should match the first entry in each of the respective **LBLOCK** tables.

As with **BCINT1**, the search routine which determines the interpolation stencil assumes that the mating grid lines are piecewise linear approximations to the same curve in the interpolation direction. A global search is performed for the proper mating cell of the first index. The closest cell to the point of interest is taken as the mating cell. A localized search is performed for the mating cells of the remaining points. The local search starts at the mating cell of the preceding point and searches along the mating boundary until the mating cell containing the new point is found. In the event that the mating cell is not found before the upper limit is reached in the mating block, the search continues from the lower limit in the mating block. This implies two things: the physical domain of the interpolation must be the same in the two blocks, and the domain is assumed to be periodic if the search routine goes past an endpoint.

Restrictions/Limitations

The **BCINTM** boundary specification is restricted to mesh interfaces which lie on a common surface (no significant overlap). Generally, endpoints of the interpolated region in the two blocks should be coincident. As with **BCINT1**, there is at least one exception to this rule based on the above description of the search routine. In the case of concentric O-grids, the endpoints of the two blocks may be misaligned (see the **BCINT1** description for details). The interpolation routine will find the appropriate stencil for each point because the grids are periodic.

The **BCINTM** condition reduces to a **PATCH** condition if the mating blocks are actually contiguous. However, due to the linear interpolation used in **BCINTM**, the scheme does not maintain either global or local conservation of flow variables across a non-contiguous mesh interface.

The **BCINTM** condition is the only non-contiguous patching routine for multiple blocks. The **BCINTM** condition will run in either serial or parallel *ADPAC08* calculations.

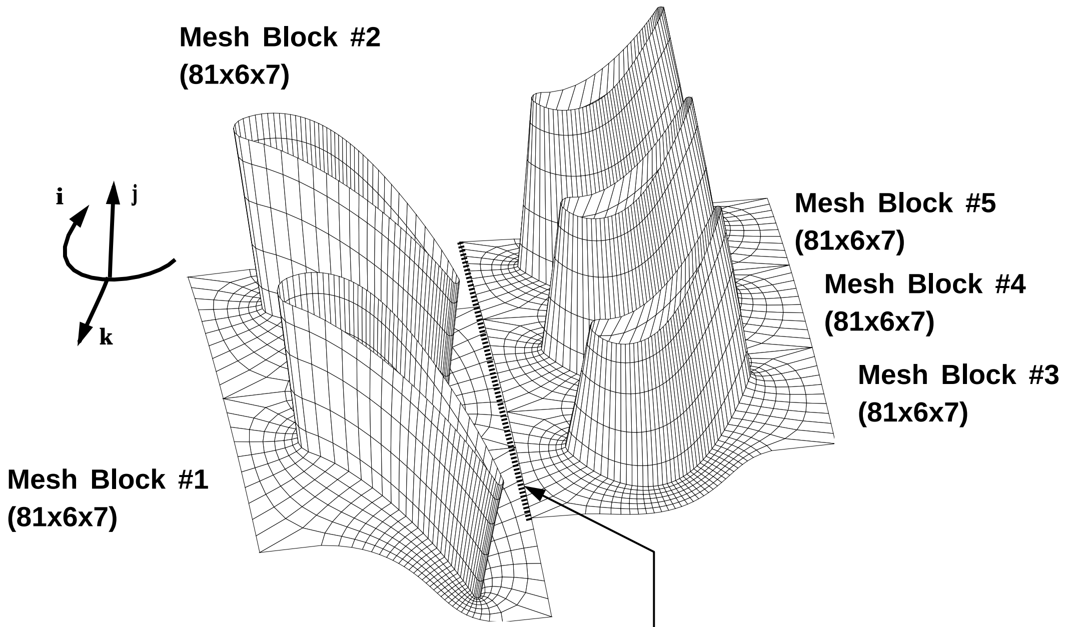
Common Errors

- Failure to provide 2 **BCINTM** statements for each interface.
- Incorrectly specified or misaligned extents of boundary regions (values of **M1LIM1, M1LIM2, N1LIM1, N1LIM2, M2LIM1, M2LIM2, N2LIM1, N2LIM2** do not correctly define the interface extents on blocks **LBLOCK1** and **LBLOCK2**).
- Incorrectly specified index for the interpolation direction for **LBLOCK1** or **LBLOCK2**.
- Attempt to use **BCINTM** for a boundary which has 2 misaligned coordinates.
- Attempt to use **BCINTM** for boundaries which are not monotonic along the interpolated index.

- Incorrect ordering of the **LBLOCK2** table of data.
- Attempt to use **BCINTM** for interfaces with multiple interpolation directions on the same side of the interface.
- Attempt to use **BCINTM** for interfaces with multiple **LFACE** or **LDIR** requirements in the **LBLOCK1** table of data.

BCPRM

Boundary Condition Procedure for Patched Relatively Rotating Mesh Blocks with Multiple Row 1 Patches



Relatively Rotating Mesh Block Interface Between Grids in Adjacent Blade Rows Can Employ a **BCPRM** Specification (illustrated in Boundary Data File Format statements below)

Application

The **BCPRM** specification is used in application involving neighboring relatively rotating mesh blocks, such as in rotor/stator interaction problems.

Boundary Data File Format:

The boundary data file specifications for the mesh interface indicated in the illustrative graphic for the **BCPRM** boundary condition and a simple outline of the mesh topography are given below. Note that blocks 1 and 2 require multiple **BCPRM** entries in the data tables due to the location of the O-grid cut line. The topography below depicts a multiple passage 3-D O-grid system for a turbine stage.

```
                  I-K Plane
                   K=KLAST
----------------------------
|     ____                 |----------------
|    /    \                |        _      |
|   <____  \      I=ILAST  |       / > I=ILAST
|    k=1 \  \--------------|      / /------|
|         \  \    I=1      |     / /  I=1  |
|           \ \            |   -/ /        |
|Block #2     \>           |   \_/ Block #5|
|                          |_______________|
----------------------------        _      |
|     ____                 |       / > I=ILAST
|    /    \                |      / /------|
|   <____  \      I=ILAST  |     / /  I=1  |
|    k=1 \  \--------------|   -/ /        |
|         \  \    I=1      |   \_/ Block #4|
|           \ \            |_______________|
|Block #1     \>           |        _      |
|                          |       / > I=ILAST
----------------------------      / /------|
                    K=KLAST|     / /  I=1  |
                           |   -/ /        |
                           |   \_/ Block #3|
                           |_______________|
                    I-J Plane

   ------------------------------------------
   |     |     A     |     |  |  A  |     |
   |     |     |     |     |  |  |  |     |
   |     |     |     |     |  |  |  |     |
   |     |<---BLADE--->|     |  |<--->|     |
   |     |    #1     |     |  | #2  |     |
   |     |     |     |     |  |  |  |     |
   |     |     |     |     |  |  |  |     |
   |     |     |     |     |  |  |  |     |
   |     |     |     |     |  |  |  |     |
   |     |     V     |     |  |  V  |     |
   ------------------------------------------

BCPRM     1  3  K  K  M  M  I  J   7   7   1   6   1   6  36  46   1   6
THPER
     0.41887903
NBCPRR - NUMBER OF BLOCKS IN LBLOCK2 TABLE OF BCPRM SPECIFICATION
    3
LBLOCK2B  LFACE2B  LDIR2B  L2LIMB  M2LIM1B  M2LIM2B  N2LIM1B  N2LIM2B
3         K        M       7       36       46       1        6
4         K        M       7       36       46       1        6
5         K        M       7       36       46       1        6
NRRDAT - NUMBER OF BLOCKS IN LBLOCK1 TABLE OF BCPRM SPECIFICATION
    4
LBLOCK1B  LFACE1B  LDIR1B  L1LIMB  M1LIM1B  M1LIM2B  N1LIM1B  N1LIM2B
1         K        M       7        1        6       1        6
1         K        M       7       76       81       1        6
2         K        M       7        1        6       1        6
2         K        M       7       76       81       1        6
BCPRM     3  1  K  K  M  M  I  J   7   7  36  46   1   6   6   1   1   6
THPER
     0.41887903
NBCPRR - NUMBER OF BLOCKS IN LBLOCK2 TABLE OF BCPRM SPECIFICATION
    4
LBLOCK2B  LFACE2B  LDIR2B  L2LIMB  M2LIM1B  M2LIM2B  N2LIM1B  N2LIM2B
```

```
1         K         M        7          6          1        1         6
1         K         M        7         81         76        1         6
2         K         M        7          6          1        1         6
2         K         M        7         81         76        1         6
NRRDAT - NUMBER OF BLOCKS IN LBLOCK1 TABLE OF BCPRM SPECIFICATION
    3
LBLOCK1B  LFACE1B  LDIR1B  L1LIMB  M1LIM1B  M1LIM2B  N1LIM1B  N1LIM2B
3         K         M        7         36         46        1         6
4         K         M        7         36         46        1         6
5         K         M        7         36         46        1         6
```

Note that a complete **BCPRM** specification generally requires at least two **BCPRM** statement lines in the boundary data file. In the example above, the first specification provides the interblock communication for the meshes representing blade row 1 from the meshes representing blade row 2, and the second specification provides the communication for the meshes representing blade row 2 from the meshes representing blade row 1. It is a common error to underspecify a **BCPRM** boundary by only providing a single line per interface.

Description:

The **BCPRM** statement is an extension of the **BCPRR** statement to include the specification of multiple **LBLOCK1** patches. As with **BCPRR**, the the **BCPRM** statement specifies that a time-space interpolation utilizing data from several neighboring mesh blocks is to be performed to determine the boundary data for the **LBLOCK1** patches. See the discussion of **BCPRR** for details about specifying the **LBLOCK2** table of data, and restrictions on the use of **BCPRM**. **BCPRM** differs from **BCPRR** only in the following way: an additional table of values allows multiple **LBLOCK1** patches to be specified. One advantage of **BCPRM** is clearly visible in the above example: only two boundary specifications are required to patch the two blade rows together, compared to seven specifications using **BCPRR**. Another, less obvious advantage is that **BCPRM** executes much faster than **BCPRR** in a parallel computing environment. Any **BCPRM** specification can be equally represented as a series of **BCPRR** specifications. The additional table of data associated with the **LBLOCK1** patches in a **BCPRM** statement is essentially the same as the table for the **LBLOCK2** patches (see the description of **BCPRR** for additional details. A comment line is followed by a line containing the number of patches in the **LBLOCK1** row. Another comment line is followed by the specification of the limits on each **LBLOCK1** patch. One restriction on the use of **BCPRM** is that all of the **LBLOCK1** patches must share a common **LFACE** and **LDIR**. This requirement can be met by the use of multiple **BCPRM** or **BCPRR** specifications.

Restrictions/Limitations

The **BCPRM** boundary specification is restricted to mesh interfaces which lie on a common surface (no significant overlap), and have common axial and radial mesh coordinates. The mesh must satisfy the coordinate restrictions listed in the table above. The **LBLOCK1** table of patches must share a common face and direction as noted above. The **BCPRM** procedure is only applicable to 3-D mesh systems.

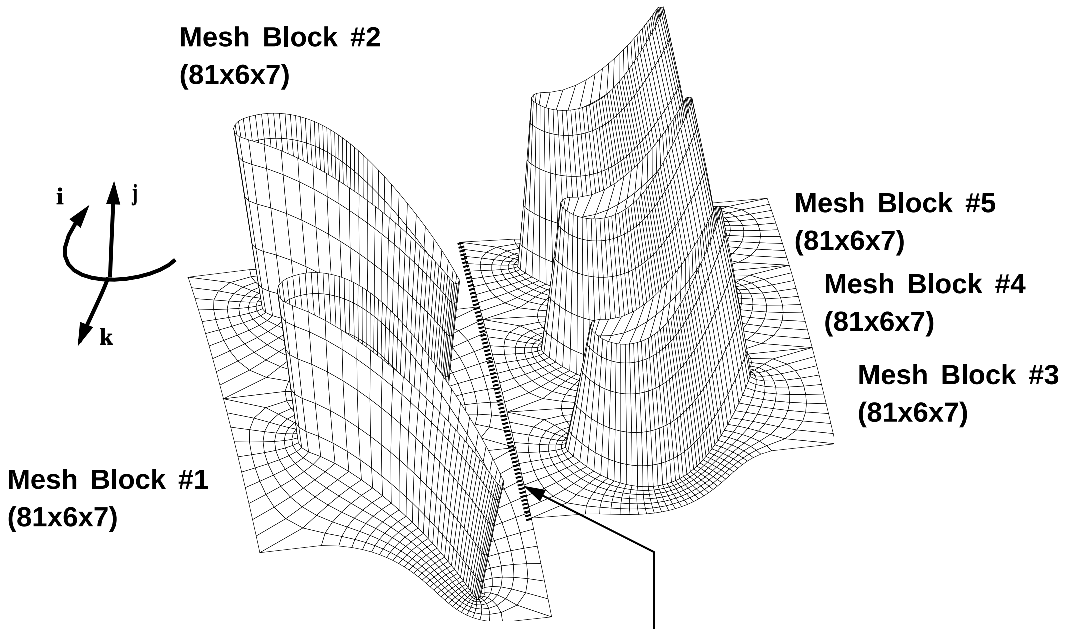
Common Errors

- Failure to provide 2 **BCPRM** statements for each interface
- Incorrectly specified or misaligned extents of boundary regions (values of **M1LIM1, M1LIM2, N1LIM1, N1LIM2, M2LIM1B, M2LIM2B, N2LIM1B, N2LIM2B** do not correctly define the interface extents on blocks **LBLOCK1** and **LBLOCK2B**)
- Attempt to use **BCPRM** on a 2-D mesh block.
- Attempt to use **BCPRM** at an interface between two Cartesian solution meshes.
- Meshes do not satisfy coordinate restrictions listed above.
- Meshes have dissimilar axial and radial coordinates at the interface.

- Neighboring blade row 1 segments not listed in increasing theta coordinate.
- Application of **BCPRM** to mesh interfaces which do not share a common surface, or which have excess overlap.

BCPRR

Boundary Condition Procedure for Patched Relatively Rotating Mesh Blocks



Relatively Rotating Mesh Block Interface Between Grids in Adjacent Blade Rows Requires a **BCPRR** Specification (illustrated in Boundary Data File Format statements below)

Application

The **BCPRR** specification is used in application involving neighboring relatively rotating mesh blocks, such as in rotor/stator interaction problems.

Boundary Data File Format:

The boundary data file specifications for the mesh interface indicated in the illustrative graphic for the **BCPRR** boundary condition and a simple outline of the mesh topography are given below. Note that blocks 1 and 2 require multiple **BCPRR** specifications due to the location of the O-grid cut line.

```
The geometry is for multiple blade row
multiple passage capped 3-D O-Grid system.

                    I-K Plane
                     K=KLAST
   ---------------------------
   |     ____                |---------------
   |    /    \               |       _      |
   |   <____ \      I=ILAST  |      / > I=ILAST
   |    k=1 \  \-------------|     / /------|
   |         \  \   I=1      |    / /  I=1  |
   |           \ \           |  -/ /        |
   |Block #2     \>          |  \_/ Block #5|
   |                         |______________|
   ---------------------------       _      |
   |     ____                |      / > I=ILAST
   |    /    \               |     / /------|
   |   <____ \      I=ILAST  |    / /  I=1  |
   |    k=1 \  \-------------|  -/ /        |
   |         \  \   I=1      |  \_/ Block #4|
   |           \ \           |______________|
   |Block #1     \>          |       _      |
   |                         |      / > I=ILAST
   ---------------------------     / /------|
                      K=KLAST|    / /  I=1  |
                             |  -/ /        |
                             |  \_/ Block #3|
                             |______________|
                    I-J Plane

   ------------------------------------------
   |     |      A      |     |  |  A  |     |
   |     |      |      |     |  |  |  |     |
   |     |      |      |     |  |  |  |     |
   |     |<---BLADE--->|     |  |<--->|     |
   |     |     #1      |     |  | #2  |     |
   |     |      |      |     |  |  |  |     |
   |     |      |      |     |  |  |  |     |
   |     |      |      |     |  |  |  |     |
   |     |      |      |     |  |  |  |     |
   |     |      V      |     |  |  V  |     |
   ------------------------------------------

BCPRR     1  3  K  K  M  M  I  J   7   7   1   6   1   6  36  46   1   6
THPER
     0.41887903
NBCPRR - NUMBER OF BLOCKS IN BCPRR SPECIFICATION
    3
LBLOCK2B  LFACE2B  LDIR2B  L2LIMB  M2LIM1B  M2LIM2B  N2LIM1B  N2LIM2B
3         K        M       7       36       46       1        6
4         K        M       7       36       46       1        6
5         K        M       7       36       46       1        6
BCPRR     1  3  K  K  M  M  I  J   7   7  76  81   1   6  36  46   1   6
THPER
     0.41887903
NBCPRR - NUMBER OF BLOCKS IN BCPRR SPECIFICATION
    3
LBLOCK2B  LFACE2B  LDIR2B  L2LIMB  M2LIM1B  M2LIM2B  N2LIM1B  N2LIM2B
3         K        M       7       36       46       1        6
4         K        M       7       36       46       1        6
5         K        M       7       36       46       1        6
BCPRR     2  3  K  K  M  M  I  J   7   7   1   6   1   6  36  46   1   6
THPER
     0.41887903
NBCPRR - NUMBER OF BLOCKS IN BCPRR SPECIFICATION
```

```
    3
LBLOCK2B  LFACE2B  LDIR2B  L2LIMB  M2LIM1B  M2LIM2B  N2LIM1B  N2LIM2B
3         K        M       7       36       46       1        6
4         K        M       7       36       46       1        6
5         K        M       7       36       46       1        6
BCPRR     2  3  K  K  M  M  I  J   7   7  76  81   1   6  36  46   1   6
THPER
     0.41887903
NBCPRR - NUMBER OF BLOCKS IN BCPRR SPECIFICATION
    3
LBLOCK2B  LFACE2B  LDIR2B  L2LIMB  M2LIM1B  M2LIM2B  N2LIM1B  N2LIM2B
3         K        M       7       36       46       1        6
4         K        M       7       36       46       1        6
5         K        M       7       36       46       1        6
BCPRR     3  1  K  K  M  M  I  J   7   7  36  46   1   6   6   1   1   6
THPER
     0.41887903
NBCPRR - NUMBER OF BLOCKS IN BCPRR SPECIFICATION
    4
LBLOCK2B  LFACE2B  LDIR2B  L2LIMB  M2LIM1B  M2LIM2B  N2LIM1B  N2LIM2B
1         K        M       7        6        1       1        6
1         K        M       7       81       76       1        6
2         K        M       7        6        1       1        6
2         K        M       7       81       76       1        6
BCPRR     4  1  K  K  M  M  I  J   7   7  36  46   1   6   6   1   1   6
THPER
     0.41887903
NBCPRR - NUMBER OF BLOCKS IN BCPRR SPECIFICATION
    4
LBLOCK2B  LFACE2B  LDIR2B  L2LIMB  M2LIM1B  M2LIM2B  N2LIM1B  N2LIM2B
1         K        M       7        6        1       1        6
1         K        M       7       81       76       1        6
2         K        M       7        6        1       1        6
2         K        M       7       81       76       1        6
BCPRR     5  1  K  K  M  M  I  J   7   7  36  46   1   6   6   1   1   6
THPER
     0.41887903
```

```
NBCPRR - NUMBER OF BLOCKS IN BCPRR SPECIFICATION
    4
LBLOCK2B  LFACE2B  LDIR2B  L2LIMB  M2LIM1B  M2LIM2B  N2LIM1B  N2LIM2B
1         K        M       7        6        1       1        6
1         K        M       7       81       76       1        6
2         K        M       7        6        1       1        6
2         K        M       7       81       76       1        6
```

Note that a complete **BCPRR** specification generally requires at least two **BCPRR** statement lines in the boundary data file. In the example above, the first four specification provides the interblock communication for the meshes representing blade row 1 from the meshes representing blade row 2, and the final three specifications provides the communication for the meshes representing blade row 2 from the meshes representing blade row 1. It is a common error to underspecify a **BCPRR** boundary by only providing a single line per interface.

Description:

The **BCPRR** statement specifies that a time-space interpolation utilizing data from several neighboring mesh blocks is to be performed to determine the boundary data for block **LBLOCK1**. This time-space interpolation provides the computational means of performing time-dependent predictions of the flow through multiple blade row turbomachines (see the discussion in Section 2.2). In order to perform this type of calculation, several conditions must be satisfied. For calculations involving blade rows with dissimilar blade counts, it is necessary to model several blade passages per blade row. The number of blade passages modeled should be chosen such that the overall circumferential span of each blade row is identical. This implies that the blade counts should be reducible to simple integer ratios (1:2, 3:4, etc.) to avoid the need for modeling an excessive number of blade passages. For example, in the illustrative graphic above, if we seek a solution for a single stage turbomachine involving two blade rows with blade counts of 30 and 45, respectively (reduced blade ratio of 2:3), then the simulation would require 2 blade passages for the first blade row and 3 passages from the second blade row, such that the overall circumferential pitch for either blade row is $\frac{2\pi}{15}$ (the number

15 chosen as the largest common factor in the blade counts 30 and 45). The second restriction is that the interface separating two adjacent blade rows be a surface of revolution, and that meshes along this interface have common axial and radial grid distributions. This restriction simplifies the time-space interpolation provided by the **BCPRR** specification. This boundary condition requires the specification of additional data, as shown in the format descriptor above. The variable following the label **THPER** defines the total circumferential span of the neighboring blade row's mesh representation in radians. For example, using the blade counts given in the previous example, the circumferential span represented in each blade row is determined by $\frac{2\pi}{15}$, and therefore **THPER** should be 0.41887903. The variable following the next label, **NBCPRR**, indicates the number of mesh blocks through which the time-space interpolation is to be performed. In the example above, if we are applying the **BCPRR** specification to the first blade row, then **NBCPRR** should be 3, since there are 3 mesh block surfaces in the neighboring blade row defining the circumferential extent of relative motion of the first blade row. The numbers immediately following the labels **LBLOCK2B**, **LFACE2B**, **LDIR2B**, **L2LIMB**, **M2LIM1B**, **M2LIM2N**, **N2LIM1B**, and **N2LIM2B** represent the values of **LBLOCK2**, **LFACE2**, **LDIR2**, **L2LIM**, **M2LIM1**, **M2LIM2**, **N2LIM1**, and **N2LIM2** (see the beginning of this section for an explanation of these variables) for *each* of the individual **NBCPRR** segments used in the construction of the circumferential data array. The **NBCPRR** segments and their respective circumferential direction indices (either **M2LIM1B**, **M2LIM2B** or **N2LIM1B**,**N2LIM2B** *must* be listed in order of increasing theta coordinate. Due to the complex nature of the circumferential interpolation operator, this boundary condition is restricted to specific mesh configurations. The following chart describes the permitted mesh configurations for the **BCPRR** specification:

BCPRR Boundary Specification Mesh Coordinate Restrictions

LFACE1 (Block #1 Face)	LFACE2 (Block #2 Face)	Circumferential Coordinate Direction	Grids Must be Aligned in this Coordinate
I	I or K	K or I	J

`J`	`J only`	`K`	`I`
`K`	`I or K`	`K or I`	`J`

In the example described above, if block numbers 1 and 2 are the block numbers for the first blade row, and block numbers 3, 4, and 5 are the block numbers for the second blade row, then the **BCPRR** specification for each of the first blade row blocks would set **THPER** = 0.41887903, **NBCPRR** = 2, and **LBLOCK2B** = 3, 4, 5. In a similar manner, the specification for each of the blocks in the second blade row would set **THPER** = 0.41887903, **NBCPRR** = 4 (due to the use of the O-type mesh for each airfoil, the extent of the interface between the two blade rows requires 2 mesh surfaces from each of the blade row 1 airfoil meshes), and **LBLOCK2B** = 1, 1, 2, 2. It should be mentioned that this specification is somewhat unique in that more than one block is involved in the boundary specification, therefore the variable **LBLOCK2** is essentially ignored; however, since the blocks specified by the **LBLOCK2B** variable are assumed to be essentially duplicate representations of neighboring blade passages, the variables **L2LIM, M2LIM1, M2LIM2, N2LIM1,** and **N2LIM2** are also ignored. The time-space interpolation is constructed to permit the relative rotation of blocks representing neighboring blade rows and therfore cannot be applied to Cartesian solution meshes. The simulation is initiated from the relative position of the blocks at the start of the calculation t=0. The interpolation scheme is area weighted to maintain a conservative property across the interface between the relatively rotating mesh blocks (see the Final Report for additional details on the implementation of this boundary procedure).

Restrictions/Limitations

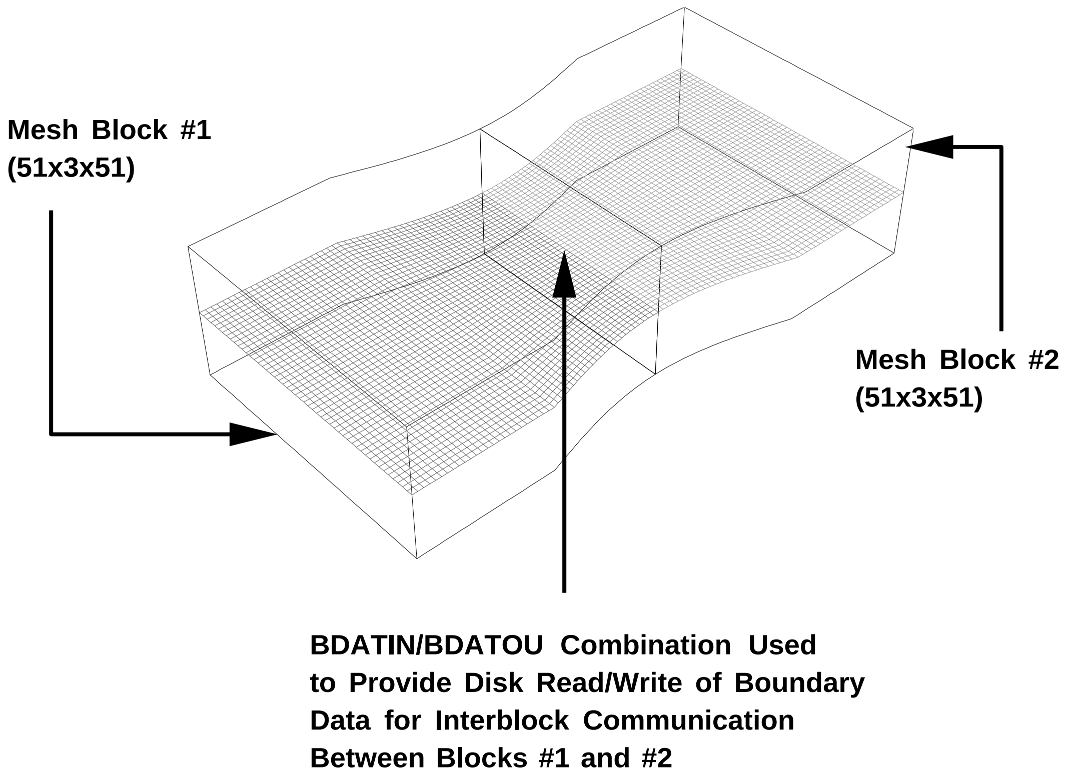
The **BCPRR** boundary specification is restricted to mesh interfaces which lie on a common surface (no significant overlap), and have common axial and radial mesh coordinates. The mesh must satisfy the coordinate restrictions listed in the table above. The **BCPRR** procedure is only applicable to 3-D mesh systems.

Common Errors

- Failure to provide 2 **BCPRR** statements for each interface
- Incorrectly specified or misaligned extents of boundary regions (values of **M1LIM1, M1LIM2, N1LIM1, N1LIM2, M2LIM1B, M2LIM2B, N2LIM1B, N2LIM2B** do not correctly define the interface extents on blocks **LBLOCK1** and **LBLOCK2B**)
- Attempt to use **BCPRR** on a 2-D mesh block.
- Attempt to use **BCPRR** on a Cartesian solution mesh.
- Meshes do not satisfy coordinate restrictions listed above.
- Meshes have dissimilar axial and radial coordinates at the interface.
- Neighboring blade row segments not listed in increasing theta coordinate.
- Application of **BCPRR** to mesh interfaces which do not share a common surface, or which have excess overlap.
- **BCPRR** runs very slow on multiple processors - use **BCPRM** instead.

BDATIN

File Read In Mesh Interface Patching Scheme


Mesh Block #1
(51x3x51)
Mesh Block #2
(51x3x51)
BDATIN/BDATOU Combination Used
to Provide Disk Read/Write of Boundary
Data for Interblock Communication
Between Blocks #1 and #2

Application

The **BDATIN** specification is used to read in boundary data from an external file. This file may be either be created by an external program, or by the *ADPAC08* boundary specification **BDATOU**. The application illustrated above indicates an application of the **BDATIN/BDATOU** combination for a two block nozzle flow case. The **BDATIN/BDATOU** combination is applied to the interface between the two mesh blocks and is equivalent to a **PATCH** specification, except that the interblock communication is accomplished through disk read/write rather than shared memory communication.

Boundary Data File Format

The boundary data file specifications for the mesh interface indicated in the illustrative graphic for the **BDATIN** boundary condition are given below:

```
BDATIN 1 2 I I M P J K  51   1   1   3  1 51   1   3  1 51
FILENAME
bc.12.data
BDATIN 2 1 I I P M J K   1  51   1   3  1 51   1   3  1 51
FILENAME
bc.21.data
```

Note that a complete **BDATIN** specification requires the specification of a filename from which the boundary data is read.

Description

The **BDATIN** statement is utilized to provide boundary data for a mesh surface through external file specification. During the application of a **BDATIN** specification, an external file is opened, and phantom cell boundary data are read in for the appropriate computational cells. The external file data may be created by an external program, or through the application of a **BDATOU** specification. A coupled set of **BDATIN/BDATOU** specifications can be effectively used to replace a **PATCH** boundary specification. In this case, interblock communication would be achieved through external file read/write rather than shared memory. If the **BDATIN/BDATOU** combination is used to replace an equivalent **PATCH** condition, it should be noted that both the **BDATIN** and **BDATOU** specifications should be written in the same manner as the **PATCH** statement. In other words, the **BDATIN** data is read in to the **LBLOCK1** block on the mesh cells defined by **L1LIM, M1LIM1, M1LIM2, N1LIM1** and **N1LIM2**, and the

BDATOU data is written out from the **LBLOCK2** block on the mesh cells defined by **L2LIM, M2LIM1, M2LIM2, N2LIM1** and **N1LIM2**. The **BDATIN/BDATOU** routines were developed in conjunction with early parallelization studies for the *ADPAC08* to permit interblock communication via shared disk file read/write operations. The routines are now considered useful for coupling the *ADPAC08* code with other codes capable of providing or using specified boundary data.

A **BDATIN** specification requires two additional lines in addition to the normal boundary data file descriptor, as shown above. The first additional line is simply a label, while the second line indicates the file name relative to the current directory from which data will be read in for this particular boundary condition.

Restrictions/Limitations

The **BDATIN/BDATOU** coupling scheme is restricted to mesh interfaces which have a one to one mesh point correspondance. Other restrictions appropriate for the **PATCH** boundary condition also apply to mesh coupling using the **BDATIN/BDATOU** scheme. Data provided in the external file for the **BDATIN** specification must represent cell centered data and must be normalized consistently with the *ADPAC08* flow variable nondimensionalization procedure.

Common Errors

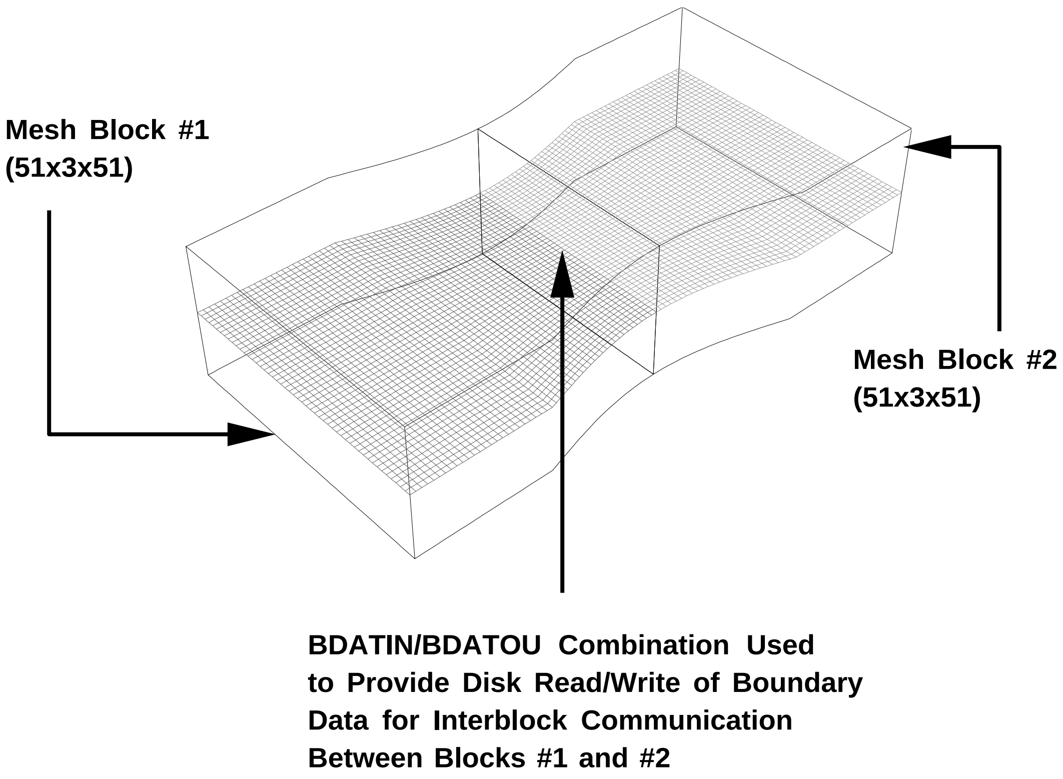
- Failure to provide file name for **BDATIN** boundary data file specification.
- Incorrectly specified or misaligned extents of boundary regions (values of **M1LIM1, M1LIM2, N1LIM1, N1LIM2, M2LIM1, M2LIM2, N2LIM1, N2LIM2** do not correctly define the interface extents on blocks **LBLOCK1** and **LBLOCK2**).
- **BDATIN/BDATOU** coupling scheme desired, but only one of the **BDATIN/BDATOU** specifications provided.

- **BDATIN/BDATOU** coupling scheme boundary specification for a periodic boundary is applied to a nonperiodic mesh.

- **BDATIN/BDATOU** coupling scheme boundary specification applied to a spatially periodic Cartesian geometry using the cylindrical coordinate solution scheme or vice versa (results in incorrect spatial periodicity relationships) The **BDATIN/BDATOU** coupling scheme boundary specifications for Cartesian geometries must use the Cartesian solution algorithm in *ADPAC08* (see input variable **FCART**).

BDATOU

File Write Out Mesh Interface Patching Scheme


Mesh Block #1
(51x3x51)
Mesh Block #2
(51x3x51)
BDATIN/BDATOU Combination Used
to Provide Disk Read/Write of Boundary
Data for Interblock Communication
Between Blocks #1 and #2

Application

The **BDATOU** specification is used to write out boundary data to an external file. This file may either be utilized by an external program, or by the *ADPAC08* boundary specification **BDATIN**. The application illustrated above indicates an application of the **BDATIN/BDATOU** combination for a two block nozzle flow case. The **BDATIN/BDATOU** combination is applied to the interface between the two mesh blocks and is equivalent to a **PATCH** specification, except that the interblock communication is accomplished through disk read/write rather than shared memory communication.

Boundary Data File Format

The boundary data file specifications for the mesh interface indicated in the illustrative graphic for the **BDATOU** boundary condition are given below:

```
BDATOU 1 2 I I M P J K  51   1   1   3  1 51   1   3  1 51
FILENAME
bc.12.data
BDATOU 2 1 I I P M J K   1  51   1   3  1 51   1   3  1 51
FILENAME
bc.21.data
```

Note that a complete **BDATOU** specification requires the specification of a filename from which the boundary data is read.

Description

The **BDATOU** statement is utilized to export boundary data for a mesh surface through external file specification. During the application of a **BDATOU** specification, an external file is opened, and near boundary cell-centered data are written in for the appropriate computational cells. The external file data may then be utilized by an external program, or through the application of a **BDATIN** specification. A coupled set of **BDATIN/BDATOU** specifications can be effectively used to replace a **PATCH** boundary specification. In this case, interblock communication would be achieved through external file read/write rather than shared memory. If the **BDATIN/BDATOU** combination is used to replace an equivalent **PATCH** condition, it should be noted that both the **BDATIN** and **BDATOU** specifications should be written in the same manner as the **PATCH** statement. In other words, the **BDATIN** data is read in to the **LBLOCK1** block on the mesh cells defined by **L1LIM, M1LIM1, M1LIM2, N1LIM1** and **N1LIM2**, and the

BDATOU data is written out to the **LBLOCK2** block on the mesh cells defined by **L2LIM, M2LIM1, M2LIM2, N2LIM1** and **N1LIM2**. The **BDATIN/BDATOU** routines were developed in conjunction with early parallelization studies for the *ADPAC08* to permit interblock communication via shared disk file read/write operations. The routines are now considered useful for coupling the *ADPAC08* code with other codes capable of providing or using specified boundary data.

A **BDATOU** specification requires two additional lines in addition to the normal boundary data file descriptor, as shown above. The first additional line is simply a label, while the second line indicates the file name relative to the current directory to which data will be written out for this particular boundary condition.

Restrictions/Limitations

The **BDATIN/BDATOU** coupling scheme is restricted to mesh interfaces which have a one to one mesh point correspondance. Other restrictions appropriate for the **PATCH** boundary condition also apply to mesh coupling using the **BDATIN/BDATOU** scheme. Data provided in the external file for the **BDATOU** specification represents near-boundary cell centered data and is normalized consistently with the *ADPAC08* flow variable nondimensionalization procedure.

Common Errors

- Failure to provide file name for **BDATOU** boundary data file specification.
- Incorrectly specified or misaligned extents of boundary regions (values of **M1LIM1, M1LIM2, N1LIM1, N1LIM2, M2LIM1, M2LIM2, N2LIM1, N2LIM2** do not correctly define the interface extents on blocks **LBLOCK1** and **LBLOCK2**).
- **BDATIN/BDATOU** coupling scheme desired, but only one of the **BDATIN/BDATOU** specifications provided.

- **BDATIN/BDATOU** coupling scheme boundary specification for a periodic boundary is applied to a nonperiodic mesh.
- **BDATIN/BDATOU** coupling scheme boundary specification applied to a spatially periodic Cartesian geometry using the cylindrical coordinate solution scheme or vice versa (results in incorrect spatial periodicity relationships) The **BDATIN/BDATOU** coupling scheme boundary specifications for Cartesian geometries must use the Cartesian solution algorithm in *ADPAC08* (see input variable **FCART**).

ENDDATA

Boundary Data File Read Terminator

Application

The **ENDDATA** statement causes the *ADPAC08* boundary data file read utility to discontinue reading lines in the boundary data file and proceeds with normal code processing. Any lines following an **ENDDATA** statement in a boundary data file are ignored.

Boundary Data File Format

The boundary data file specifications for an **ENDDATA** statement is given below:

```
ENDDATA
```

Note that the **ENDDATA** statement does not require the accompanying values of **LBLOCK1, LBLOCK2, LFACE1**, etc. as do all other boundary data file keywords.

Description

The **ENDDATA** statement is utilized to provide a terminator for the boundary data file read sequence in the *ADPAC08* code. Under normal operating conditions, the boundary data file is read in one line at a time and parsed to determine if a boundary data file keyword is present and uncommented on each line. When the end of the file is reached, the boundary data file read sequence stops, and normal processing continues as usual. In some cases, it may be desirable to terminate the boundary data file read sequence before the end of the file, and the **ENDDATA** statement is provided for this purpose. Whenever an **ENDDATA** statement is reached, the boundary data file read sequence is terminated, and all remaining lines in the boundary data file are ignored. The **ENDDATA** keyword is useful for debugging boundary condition problems, as whole portions of the boundary data file can be effectively eliminated by simply preceeding the section with an **ENDDATA** statement.

Restrictions/Limitations

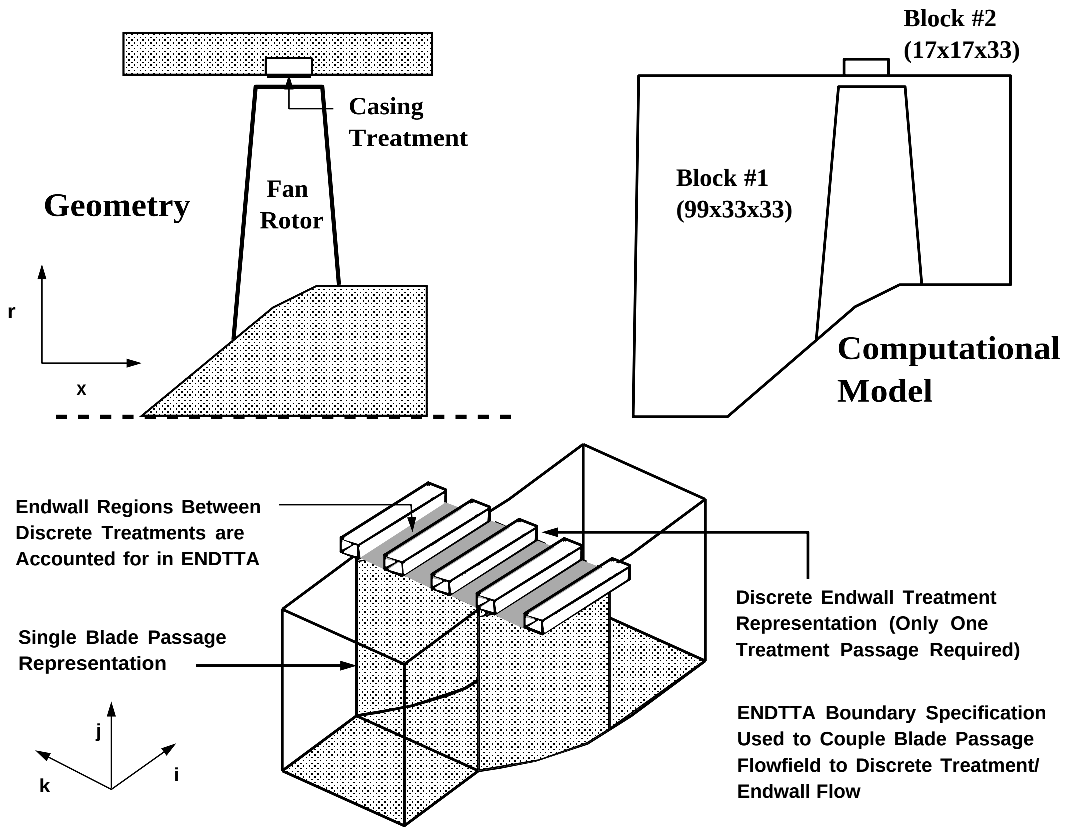
The **ENDDATA** keyword has no restrictions.

Common Errors

- Desired boundary conditions specifications following an **ENDDATA** statement are ignored.
- *ADPAC08* aborts because an insufficient number of boundary conditions were provided for the external boundaries of each mesh block (external boundaries of some mesh blocks do not have a boundary condition).

ENDTTA

Endwall Treatment Time-Average Mesh Block Interface Patching Scheme



Application

The **ENDTTA** boundary specification was developed specifically to permit numerical prediction of turbomachinery airfoil blade row flows employing endwall treatments such as slots, grooves, or embedded bladed passages in a time-averaged fashion. The example graphic above illustrates a 3-D blocked mesh system for a turbofan engine fan rotor employing an axial slot casing treatment. The **ENDTTA** boundary specification employs a time-averaging operator (circumferential average of flow variables) between adjacent rotating and nonrotating mesh blocks to simulate the effects of the blade row/endwall treatment interaction. As such, it is possible to perform steady state (representative of a time average) numerical analysis of turbomachinery blade passages and endwall treatments which have arbitrary blade passage/treatment passage count ratios.

Boundary Data File Format

The boundary data file specifications for the mesh interface indicated in the illustrative graphic for the **ENDTTA** boundary condition are given below:

```
ENDTTA    1  2  J  J  M  P  L  L  49   1  49  81   1  33   1  33   1  17
NTREAT RPM TWALL
113    0.0  0.0
```

Note that a complete **ENDTTA** specification generally requires a companion **MBCAVG** specification to complete the blade passage mesh/treatment passage mesh interface specification. In the example above, the first specification provides the interblock communication for block 1 (the blade passage mesh) to block 2 (the treatment passage mesh) which ultimately accounts for the influence of the true endwall in the boundary specification. The second specification (**MBCAVG**) is applied to the treatment passage mesh boundary to simulate the time-average (circumferential average) of the neighboring blade passage mesh. It is a common error to underspecify an **ENDTTA** boundary by only providing a single line per interface.

Description

The **ENDTTA** boundary statement provides a means for block to block communication for the prediction of the time-averaged flow for turbomachinery blade rows employing endwall treatments such as discrete slots, grooves, or embedded bladed passages. This boundary condition was developed under Task 6 of NASA Contract NAS3-25270 and theoretical details of the procedure are provided in the Final Report for Task 7 of NASA Contract NAS3-25270 [17]. The boundary condition is restricted to j=constant mesh surfaces *only* and must possess aligned coordinates in the i direction, but have misaligned mesh points and extents in the circumferential (k) direction. An example of an appropriate application of the **ENDTTA** specification is given in the illustrative graphic. The **ENDTTA** boundary specification is valid for 3-D cylindrical solution mesh blocks only.

Restrictions/Limitations

The **ENDTTA** boundary specification is restricted to mesh interfaces which lie on a common surface (no significant overlap). The **ENDTTA** procedure permits only that the k coordinates between adjacent mesh surfaces are misaligned. The **ENDTTA** procedure is only valid if applied to j=constant mesh surfaces. **ENDTTA** will not run across multiple processors in a parallel computing environment.

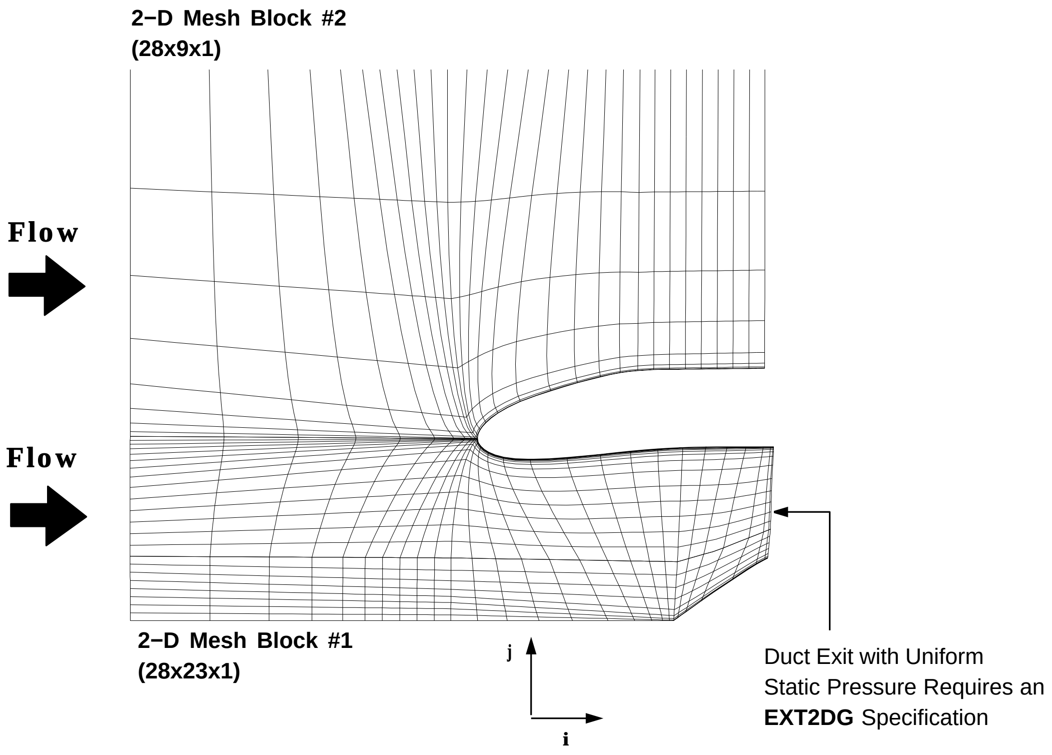
Common Errors

- Failure to provide a coupled pair of **ENDTTA** and **MBCAVG** statements for each interface.
- Failure to properly specify the values for **RPM, TWALL** and/or **NTREAT**
- Incorrectly specified or misaligned extents of boundary regions (values of **M1LIM1, M1LIM2, N1LIM1, N1LIM2, M2LIM1, M2LIM2, N2LIM1, N2LIM2** do not correctly define the interface extents on blocks **LBLOCK1** and **LBLOCK2**).
- Attempt to use **ENDTTA** for an i or k constant boundary.
- Attempt to use **ENDTTA** for a Cartesian solution mesh.

- Attempt to use **ENDTTA** for a boundary which has 2 misaligned coordinates.
- Attempt to use **ENDTTA** with multiple processors.

EXT2DG

Generic 2-D Outflow Boundary Condition



Application

The **EXT2DG** specification is used to impose a generic subsonic outflow boundary condition with a uniform exit static pressure for 2-D mesh blocks. The example graphic above illustrates a 2-D 2-block mesh system mixing two adjacent streams of varying properties. In this case, the **EXT2DG** boundary specification is used to set the outflow boundary flow properties by specifying a uniform exit static pressure. This boundary condition has been utilized extensively as an exit flow specifier for 2-D duct flows.

Boundary Data File Format

The boundary data file specifications for the exit flow mesh surfaces indicated in the illustrative graphic for the **EXT2DG** boundary condition are given below:

```
EXT2DG 1  1  I  I  M  M  J K 28 28   1  23  1   2  1  23  1   2
PEXIT
0.625
```

or the alternate specification:

```
EXT2DG 1  1  I  I  M  M  J K 28 28   1  23  1   2  1  23  1   2
PEXIT
0.625
```

Note that a complete **EXT2DG** specification requires two additional lines following the **EXT2DG** boundary data file specification line. Failure to properly specify the data in these additional lines is a common **EXT2DG** specification error.

Description

The **EXT2DG** statement specifies that a generic, subsonic, uniform static pressure exit flow boundary condition is to be applied to the mesh surface specified by **LFACE1** on the 2-D mesh block specified by **LBLOCK1**. The **EXT2DG** boundary condition should be applied for those cases where any other "specialized" exit boundary condition (such as **EXT2DT, EXT2DP**, etc.) does not apply. The **EXT2DG** boundary condition is also likely to be somewhat more efficient computationally than the other exit boundary condition procedures, at the expense of some physical simplification. The **EXT2DG** procedure utilizes a Reimann invariant formulation to compute exit velocities based on a specified constant exit static pressure. Included in the **EXT2DG** procedure is a special correction scheme which forces the flow to pass out of the flow domain at the boundary. In other words, if the computed velocities result in a local inflow at the **EXT2DG** boundary, no matter how small the magnitude of the inflow, the velocities are reset to zero at that point. This boundary condition requires the specification of additional data, as shown in the boundary data format descriptor above. The first additional line following the **EXT2DG** specification is assumed to be a label and may contain any information; however, for consistency it is recommended that the label **PEXIT** be used. The next line contains the value imposed for the variables **PEXIT** which represents the downstream exit static pressure ratio used in the **EXT2DG** characteristic solution sequence. The value of the **PEXIT** variable is the desired normalized downstream static pressure computed as:

$$PEXIT = \frac{P_{static,desired}}{P_{ref}}$$

where the variable P_{ref} is specified by the input variable **PREF**. It should be mentioned that for most geometries, the value of **PEXIT**, in combination with any inlet flow boundary conditions, will normally govern the resulting solution mass flow rate (exceptions to this rule will occur when the inlet mass flow rate boundary condition procedure **INLETM** is applied). Values of **PEXIT** <0.0 are not permitted. As the value of **PEXIT** is reduced, the flow through the boundary will ultimately choke, and further reductions of **PEXIT** will no longer increase the mass flow through the boundary. Naturally, poor convergence or solution divergence can occur if **PEXIT** is too high or too low when compared to the rest of the flow domain. In such cases where this occurs, it is recommended that the solution be started with more conservative boundary values, and then restarted using the final boundary values.

An alternate specification is provided for the **EXT2DG** boundary specification as shown in the sample application above. In this case, three values are included following the original boundary specification line. The alternate specification is provided as a means of achieving a desired mass flow rate through the bounding surface using the **EXT2DG** algorithm. The desired mass flow rate is achieved iteratively by incrementally adjusting the exit static pressure specification until the desired flow rate is achieved. Therefore, in this specification, the variable **PEXIT** described in detail above is the initial exit static pressure used in the iterative process, **EMDOT** represents the desired mass flow rate through the bounding surface in pounds mass, and **PRELAX** is a relaxation factor to stabilize the iterative process (values may range from 0.0 to 1.0, though poor convergence is likely for values larger than 0.1). This procedure is not foolproof, and suffers from the fact that when a job is restarted, if an updated exit pressure is not inserted in the boundary data file, then the pressure-mass flow iterative process will essentially start over. The *ADPAC08* code will automatically determine when to employ the iterative process by identifying the additional boundary specification variables.

Restrictions/Limitations

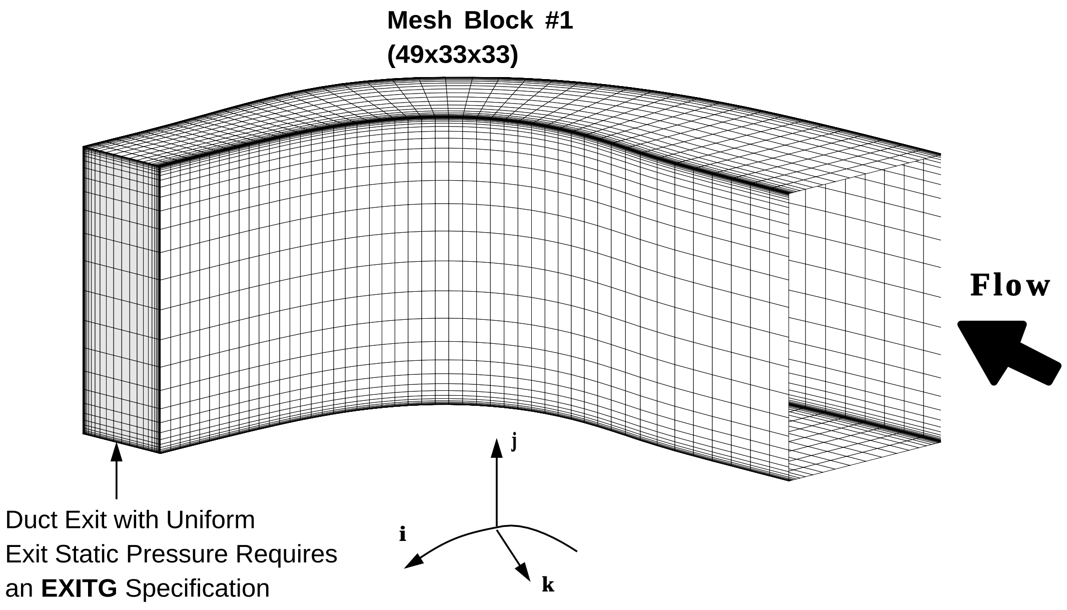
The **EXT2DG** boundary specification is restricted to 2-D mesh surfaces (3-D mesh surfaces should use the **EXITG** boundary specification).

Common Errors

- Application of **EXT2DG** to a 3-D mesh system.
- Failure to specify the additional data value **PEXIT**.
- Improper specification of thealternate (mass flow) iterative scheme.
- Reductions in the value of **PEXIT** do not increase the mass flow rate because of flow choking.
- Value of **PEXIT** is too high (flow cannot get started).

EXITG

Generic Outflow Boundary Condition



Application

The **EXITG** specification is used to impose a generic subsonic outflow boundary condition with a uniform exit static pressure. The example graphic above depicts a simple duct flow using a Cartesian-based H-grid, where the exit boundary plane is controlled by an **EXITG** specification. This boundary condition has been utilized extensively as an exit flow specifier for duct flows.

Boundary Data File Format

The boundary data file specification for the mesh surface indicated in the illustrative graphic for the **EXITG** boundary condition is given below:

```
EXITG 1  1  I  I  M  M  J K 49 49   1  33  1  33  1  33  1  33
PEXIT
0.625
```

or the alternate specification:

```
EXITG 1  1  I  I  M  M  J K 49 49   1  33  1  33  1  33  1  33
PEXIT    EMDOT   PRELAX
0.625    40.0    0.001
```

Note that a complete **EXITG** specification requires two additional lines following the **EXITG** boundary data file specification line. Failure to properly specify the data in these additional lines is a common **EXITG** specification error.

Description

The **EXITG** statement specifies that a generic, subsonic, uniform static pressure exit flow boundary condition is to be applied to the mesh surface specified by **LFACE1** on the block specified by **LBLOCK1**. The **EXITG** boundary condition should be applied for those cases where any other "specialized" exit boundary condition (such as **EXITT, EXITP**, etc.) does not apply. The **EXITG** boundary condition is also likely to be somewhat more efficient computationally than the other exit boundary condition procedures, at the expense of some physical simplification. **EXITG** may be used on any mesh face (I, J, or K constant) for either cylindrical or Cartesian-based solution schemes (see the input variable **FCART**). The **EXITG** procedure utilizes a Reimann invariant formulation to compute exit velocities based on a specified constant exit static pressure. Included in the **EXITG** procedure is a special correction scheme which forces the flow to pass out of the flow domain at the boundary. In other words, if the computed velocities result in a local inflow at the **EXITG** boundary, no matter how small the magnitude of the inflow, the velocities are reset to zero at that point. This boundary condition requires the specification of additional data, as shown in the Boundary Data Format descriptor above. The first additional line following the **EXITG** specification is assumed to be a label and may contain any information; however, for consistency it is recommended that the label **PEXIT** be used. The next line contains the value imposed for the variables **PEXIT** which represents the downstream exit static pressure ratio used in the **EXITG** characteristic solution sequence. The value of the **PEXIT** variable is the desired normalized downstream static pressure computed as:

$$PEXIT = \frac{P_{static,desired}}{P_{ref}}$$

where the variable P_{ref} is specified by the input variable **PREF**. It should be mentioned that for most geometries, the value of **PEXIT**, in combination with any inlet flow boundary conditions, will normally govern the resulting solution mass flow rate (exceptions to this rule will occur when the inlet mass flow rate boundary condition procedure **INLETM** is applied). Values of **PEXIT** <0.0 are not permitted. As the value of **EXITP** is reduced, the flow through the boundary will ultimately choke, and further reductions of **EXITP** will no longer increase the mass flow through the boundary. Naturally, poor convergence or solution divergence can occur if **PEXIT** is too high or too low when compared to the rest of the flow domain. In such cases where this occurs, it is recommended that the solution be started with more conservative boundary values, and then restarted using the final boundary values.

An alternate specification is provided for the **EXITG** boundary specification as shown in the sample application above. In this case, three values are included following the original boundary specification line. The alternate specification is provided as a means of achieving a desired mass flow rate through the bounding surface using the **EXITG** algorithm. The desired mass flow rate is achieved iteratively by incrementally adjusting the exit static pressure specification until the desired flow rate is achieved. Therefore, in this specification, the variable **PEXIT** described in detail above is the initial exit static pressure used in the iterative process, **EMDOT** represents the desired mass flow rate through the bounding surface in pounds mass, and **PRELAX** is a relaxation factor to stabilize the iterative process (values may range from 0.0 to 1.0, though poor convergence is likely for values larger than 0.1). This procedure is not foolproof, and suffers from the fact that when a job is restarted, if an updated exit pressure is not inserted in the boundary data file, then the pressure-mass flow iterative process will essentially start over. The *ADPAC08* code will automatically determine when to employ the iterative process by identifying the additional boundary specification variables.

Restrictions/Limitations

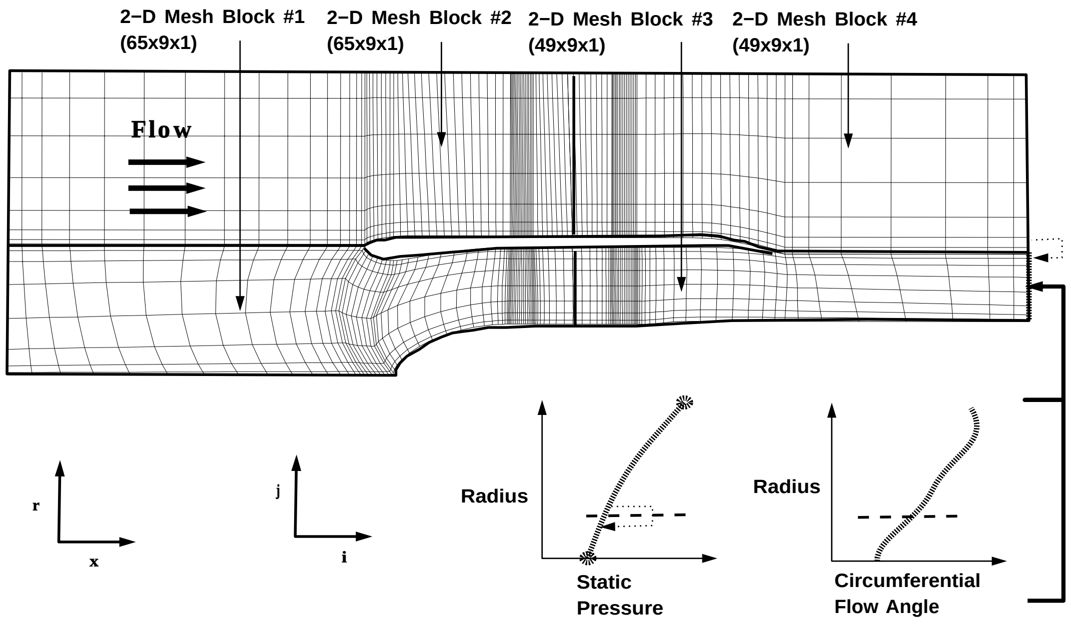
The **EXITG** boundary specification is not restricted to 3-D mesh surfaces (although for consistency 2-D mesh surfaces should use the **EXT2DG** boundary specification).

Common Errors

- Failure to specify the additional data value **PEXIT**.
- Improper specification of the alternate (mass flow) boundary scheme.
- Reductions in the value of **PEXIT** do not increase the mass flow rate because of flow choking.
- Value of **PEXIT** is too high (flow cannot get started).

EXT2DP

2-D Patched Turbomachinery Exit Boundary Condition


2−D Mesh Block #1 (65x9x1)
2−D Mesh Block #2 (65x9x1)
2−D Mesh Block #3 (49x9x1)
2−D Mesh Block #4 (49x9x1)
Flow
r
x
j
i
Radius
Static Pressure
Radius
Circumferential Flow Angle

Patched Exit Static Pressure and Radial Equilibrium for 2−D Turbomachinery Exit Flow Requires an **EXT2DP** Specification (illustrated in Boundary Data File Format statements below)

Static pressure specified at either lower or upper "j" boundary
Radial equilibrium equation integrated to complete exit static pressure specification

Application

The **EXT2DP** specification is used to impose a turbomachinery-based exit boundary condition based on radial equilibrium for 2-D mesh systems employing multiple blocks radially across the exit plane. The example graphic above illustrates a four block mesh system used to predict the axisymmtric flow through a high bypass ratio turbofan engine geometry. The solution utilizes a specified freestream static pressure at the outer boundary of block 4, and an **EXT2DT** specification to integrate the radial equilibrium equation equation inward radially along the outflow boundary. In order to continue the radial equilibrium integration process across the block boundary between blocks 3 and 4, an **EXT2DP** specification is used to patch the two blocks. This boundary condition has been utilized extensively in conjunction with the **EXT2DT** specification as an exit flow specifier for both ducted and unducted turbomachinery flows.

Boundary Data File Format

The boundary data file specification for the 2-D mesh surface indicated in the illustrative graphic for the **EXT2DP** boundary condition is given below:

```
EXT2DP 4  3  I  I  M  M  L H  49 49  1   9  1   2   9  9  1   2
```

Note that the **M2LIM1, M2LIM2** variables in the **EXT2DP** specification define a single j mesh line in mesh block **LBLOCK2**. Failure to properly regard this requirement is a common **EXT2DP** specification error. It should also be mentioned that **EXT2DP** also requires proper specification of the **LSPEC1** variable for proper execution.

Description

The **EXT2DP** keyword specifies that a turbomachinery-based radial equilibrium patched exit flow boundary condition is to be applied to the mesh surface specified by **LFACE1** on the 2-D block specified by **LBLOCK1**. The **EXT2DP** boundary condition was specifically designed as an exit flow boundary procedure for axial and mixed flow turbomachinery geometries employing multiple, stacked 2-D mesh blocks (radially) at an exit boundary plane. The **EXT2DP** boundary condition procedure utilizes a combination static pressure specification and integration of the radial equilibrium equation to define the static pressure field at all points on the boundary surface. The initial static pressure specification used to initiate the radial equilibrium integration process is obtained from a neighboring mesh block. As a result of the complexity of this procedure, several mesh restrictions were imposed to simplify the application of this approach. The primary assumption is that the integration of the radial equilibrium equation may be performed along the j coordinate direction of the mesh. Hence, the j coordinate should be the radial-like direction. A single specification of static pressure is required at either the maximum or minimum extreme of the j coordinate of the boundary surface in order to initiate the integration process. The direction of integration, and location of application of the specified exit static pressure are determined by the **LSPEC1** variable in the calling sequence. If **LSPEC1** = L, for LOW, then **PEXIT** is applied to the lower (smallest value) of the j index, and the radial equilibrium equation is integrated outward (increasing j direction). If **LSPEC1** = H, for HIGH, then **PEXIT** is applied to the upper (largest value) of the j index, and the radial equilibrium equation is integrated inward (decreasing j direction). The direction of integration implied by **LSPEC1** must be consistent with the location of the neighboring mesh block (**LBLOCK2**) from which the initial pressure data is derived. The j coordinate location from which the pressure is taken in mesh block **LBLOCK2** is determined by the **M2LIM1, M2LIM1** variable, and the specification of the **LSPEC2** control parameter. If the **M2LIM1,M2LIM2** combination is taken from the higher j index, then **LSPEC2** should be **H**. If the **M2LIM1,M2LIM2** combination is taken from the lower j index, then **LSPEC2** should be **L**. Under most circumstances, the static pressure is taken from either the uppermost or lowermost j coordinate, in which case **LSPEC2** should be either **H** or **L**, respectively. The remaining flow variables on the **EXT2DP** boundary are updated by a Reimann invariant formulation based on the resulting local static pressure field. Included in the **EXT2DP** procedure is a special correction scheme which forces the flow to pass out of the flow domain. In other words, if the computed velocities result in a local inflow at the **EXT2DP** boundary, no matter how small the magnitude of the inflow, the velocities are reset to zero at that point.

Restrictions/Limitations

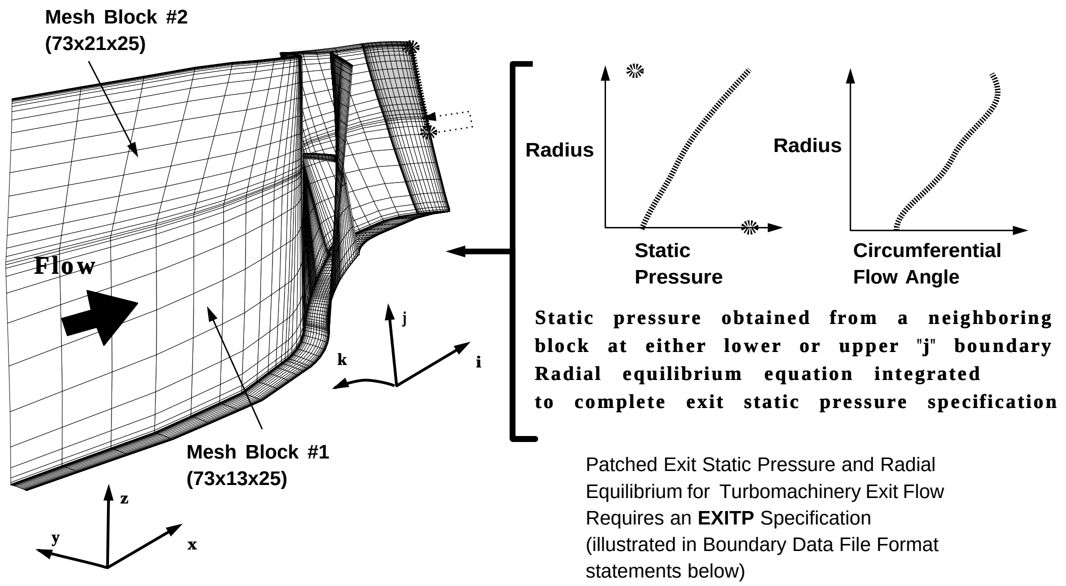
The **EXT2DP** boundary condition assumes that the mesh is oriented in such a fashion that the radial coordinate is defined as $r = \sqrt{y^2 + z^2}$. For axial flow turbomachinery, this implies that the axis of rotation (or the centerline) coincides with the x axis. It is also required that the radial-like direction of the mesh be defined by the j coordinate, and is therefore not valid on a j =constant mesh plane. This is required in order to properly integrate the radial equilibrium equation to complete the exit static pressure specification. Examples of this type of mesh system can be found in the chapter defining standard configurations. The **EXT2DP** boundary specification is restricted to 2-D mesh surfaces (3-D mesh surfaces should use the **EXITP** boundary specification). By default, it is important that this type of boundary condition be carefully specified and the final solution carefully examined to ensure that the desired mesh patching be adequately satisfied. It is a common error to patch to the wrong grid, or the wrong end of the correct grid, and still obtain a converged solution.

Common Errors

- Application of **EXT2DP** to a 3-D mesh system.
- Failure to properly specify the **LSPEC1**, **LSPEC2** variables.
- **M2LIM1** and **M2LIM2** differ.
- Radial-like direction of the mesh is not the j coordinate.
- Failure to properly specify the **LSPEC1** variable on the boundary data file specification line.
- **EXT2DP** specification patched to the wrong grid.
- **EXT2DP** specification patched to the wrong end of the correct grid.
- **EXT2DP** boundary condition used but no **EXT2DT** or **EXITT** boundary condition specified.
- **EXT2DP** boundary condition called before **EXT2DT** (not required, but could cause problems).

EXITP

Patched Turbomachinery Exit Boundary Condition


Mesh Block #2
(73x21x25)
Flow
Mesh Block #1
(73x13x25)
z
y
x
j
k
i
Radius
Static
Pressure
Radius
Circumferential
Flow Angle
Static pressure obtained from a neighboring
block at either lower or upper "j" boundary
Radial equilibrium equation integrated
to complete exit static pressure specification

Patched Exit Static Pressure and Radial Equilibrium for Turbomachinery Exit Flow Requires an **EXITP** Specification (illustrated in Boundary Data File Format statements below)

Application

The **EXITP** specification is used to impose a turbomachinery-based exit boundary condition based on radial equilibrium for mesh systems employing multiple blocks radially across the exit plane. The example graphic illustrates a two block 3-D mesh system used to predict the flow through a blade passage of a turbomachinery fan rotor with a part span shroud. The blocks are divided radially by the part span shroud, and as a result, the exit boundary plane consists of two mesh boundary segments. In order to employ a turbomachinery-based radial equilibrium exit flow boundary condition for this case, the **EXITT** specification is applied to the inner mesh block (#1) and the **EXITP** boundary condition is used for the outer block (#2) to complete the inner to outer integration of the radial equilibrium equation across the mesh block interface. This boundary condition has been utilized extensively in conjunction with the **EXITT** specification as an exit flow specifier for both ducted and unducted turbomachinery flows.

Boundary Data File Format

The boundary data file specification for the mesh surface indicated in the illustrative graphic for the **EXITP** boundary condition is given below:

```
EXITP 2   2   I   I   M   M   L L   73 73   1   21   1   25   13 13   1   25
```

Note that the **M2LIM1, M2LIM2** variables in the **EXITP** specification define a single j mesh line in mesh block **LBLOCK2**. Failure to properly regard this requirement is a common **EXITP** specification error. It should also be mentioned that **EXITP** also requires proper specification of the **LSPEC1** variable for proper execution.

Description

The **EXITP** keyword specifies that a turbomachinery-based radial equilibrium patched exit flow boundary condition is to be applied to the mesh surface specified by **LFACE1** on the block specified by **LBLOCK1**. The **EXITP** boundary condition was specifically designed as an exit flow boundary procedure for axial and mixed flow turbomachinery geometries employing multiple, stacked mesh blocks (radially) at an exit boundary plane. The **EXITP** boundary condition procedure utilizes a combination static pressure specification and integration of the radial equilibrium equation to define the static pressure field at all points on the boundary surface. The initial static pressure specification used to initiate the radial equilibrium integration process is obtained from a neighboring mesh block. As a result of the complexity of this procedure, several mesh restrictions were imposed to simplify the application of this approach. The primary assumption is that the integration of the radial equilibrium equation may be performed along the

j coordinate direction of the mesh. Hence, the j coordinate should be the radial-like direction. A single specification of static pressure is required at either the maximum or minimum extreme of the j coordinate of the boundary surface in order to initiate the integration process. The direction of integration, and location of application of the specified exit static pressure are determined by the **LSPEC1** variable in the calling sequence. If **LSPEC1** = L, for LOW, then **PEXIT** is applied to the lower (smallest value) of the j index, and the radial equilibrium equation is integrated outward (increasing j direction). If **LSPEC1** = H, for HIGH, then **PEXIT** is applied to the upper (largest value) of the j index, and the radial equilibrium equation is integrated inward (decreasing j direction). The direction of integration implied by **LSPEC1** must be consistent with the location of the neighboring mesh block (**LBLOCK2**) from which the initial pressure data is derived. The j coordinate location from which the pressure is taken in mesh block **LBLOCK2** is determined by the **M2LIM1, M2LIM1** variable, and the specification of the **LSPEC2** control parameter. If the **M2LIM1,M2LIM2** combination is taken from the higher j index, then **LSPEC2** should be **H**. If the **M2LIM1,M2LIM2** combination is taken from the lower j index, then **LSPEC2** should be **L**. Under most circumstances, the static pressure is taken from either the uppermost or lowermost j coordinate, in which case **LSPEC2** should be either **H** or **L**, respectively. The remaining flow variables on the **EXITP** boundary are updated by a Reimann invariant formulation based on the resulting local static pressure field. Included in the **EXITP** procedure is a special correction scheme which forces the flow to pass out of the flow domain. In other words, if the computed velocities result in a local inflow at the **EXITP** boundary, no matter how small the magnitude of the inflow, the velocities are reset to zero at that point.

Restrictions/Limitations

The **EXITP** boundary condition assumes that the mesh is oriented in such a fashion that the radial coordinate is defined as $r = \sqrt{y^2 + z^2}$. For axial flow turbomachinery, this implies that the axis of rotation (or the centerline) coincides with the x axis. It is also required that the radial-like direction of the mesh be defined by the j coordinate, and is therefore not valid on a

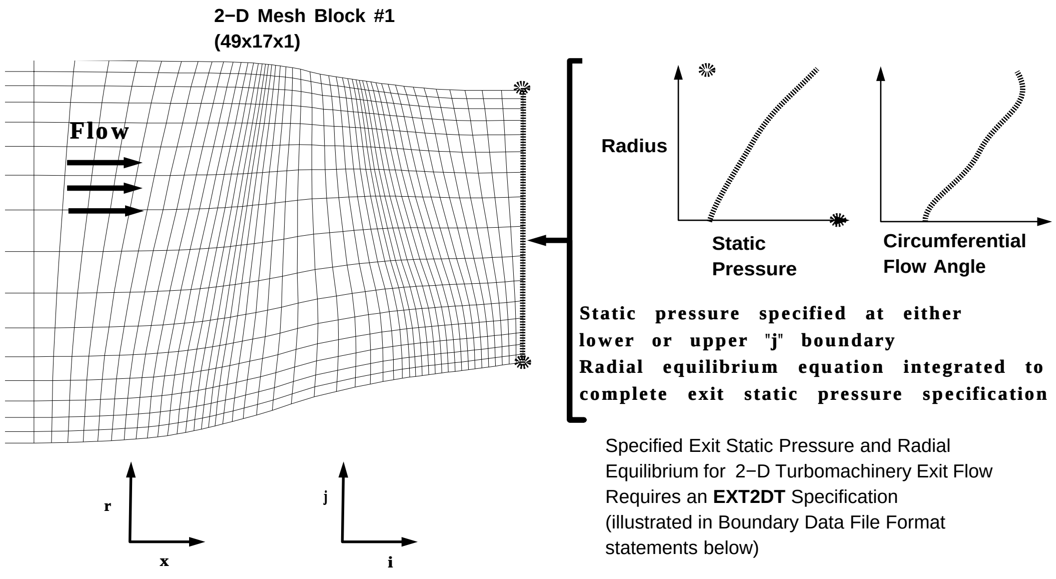
j =constant mesh plane. This is required in order to properly integrate the radial equilibrium equation to complete the exit static pressure specification. Examples of this type of mesh system can be found in the chapter defining standard configurations. The **EXITP** boundary specification is restricted to 3-D mesh surfaces (2-D mesh surfaces should use the **EXT2DP** boundary specification).

Common Errors

- Application of **EXITP** to a 2-D mesh system.
- Failure to properly specify the **LSPEC2** variable.
- **M2LIM1** and **M2LIM2** differ.
- Radial-like direction of the mesh is not the j coordinate.
- Failure to properly specify the **LSPEC1** variable on the boundary data file specification line.

EXT2DT

2-D Turbomachinery Exit Boundary Condition



2−D Mesh Block #1
(49x17x1)
Flow
Radius
Static Pressure
Circumferential Flow Angle
Static pressure specified at either lower or upper "j" boundary
Radial equilibrium equation integrated to complete exit static pressure specification
Specified Exit Static Pressure and Radial Equilibrium for 2−D Turbomachinery Exit Flow Requires an **EXT2DT** Specification (illustrated in Boundary Data File Format statements below)
r
x
j
i

Application

The **EXT2DT** specification is used to impose a turbomachinery-based exit boundary condition based on radial equilibrium for 2-D mesh blocks. The example graphic illustrated above depicts an **EXT2DT** specification for a 2-D (axisymmetric) flow solution for a turbomachinery blade row. This boundary condition has been utilized extensively as an exit flow specifier for both ducted and unducted 2-D turbomachinery flows.

Boundary Data File Format

The boundary data file specification for the mesh surface indicated in the illustrative graphic for the **EXT2DT** boundary condition is given below:

```
EXT2DT 1  1  I  I  M  M  L L  49 49  1  17  1   2  1  17  1   2
PEXIT
1.105
```

or the alternate specification:

```
EXT2DT 1  1  I  I  M  M  L L  49 49  1  17  1   2  1  17  1   2
PEXIT  EMDOT   PRELAX
1.105  13.7    0.001
```

Note that a complete **EXT2DT** specification requires two additional lines following the **EXT2DT** boundary data file specification line. Failure to properly specify the data in these additional lines is a common **EXT2DT** specification error. It should also be mentioned that **EXT2DT** also requires proper specification of the **LSPEC1** variable for proper execution.

Description

The **EXT2DT** keyword specifies that a turbomachinery-based radial equilibrium exit flow boundary condition is to be applied to the mesh surface specified by **LFACE1** on the 2-D mesh block specified by **LBLOCK1**. The **EXT2DT** boundary condition was specifically designed as an exit flow boundary procedure for 2-D axial and mixed flow turbomachinery geometries. Pure radial flow turbomachinery exit flow boundaries may usually be specified by the **EXT2DG** boundary condition. Due to the form of the radial equilibrium equation utilized in the **EXT2DG** routine, only cylindrical coordinate solution meshes are permitted to use this routine. The **EXT2DT** boundary condition procedure utilizes a combination static pressure specification and integration of the radial equilibrium equation to define the static pressure field at all points on the boundary surface. As a result of the complexity of this procedure, several mesh restrictions were imposed to simplify the application of this approach. The primary assumption is that the integration of the radial equilibrium equation may be performed along the j coordinate direction of the mesh. Hence, the j coordinate should be the radial-like direction. A single specification of static pressure is required at either the maximum or minimum extreme of the j coordinate of the boundary surface in order to initiate the integration process. The direction of integration, and location of application of the specified exit static pressure are determined by the **LSPEC1** variable in the calling sequence. If **LSPEC1** = L, for LOW, then **PEXIT** is applied to the lower (smallest value) of the j index, and the radial equilibrium equation is integrated outward (increasing j direction). If **LSPEC1** = H, for HIGH, then **PEXIT** is applied to the upper (largest value) of the j index, and the radial equilibrium equation is integrated inward (decreasing j direction). The remaining flow variables on the **EXT2DT** boundary are updated by a Reimann invariant formulation based on the resulting local static pressure field. Included in the **EXT2DT** procedure is a special correction scheme which forces the flow to pass out of the flow domain. In other words, if the computed velocities result in a local inflow at the **EXT2DT** boundary, no matter how small the magnitude of the inflow, the velocities are reset to zero at that point. This boundary condition requires the specification of additional data, as shown in the boundary data format descriptor above. The first additional line following the **EXT2DT** specification is assumed to be a label and may contain any information; however, for consistency it is recommended that the label **PEXIT** be used. The line following the **PEXIT** label contains the value of specified nondimensional exit static pressure used to initiate the radial equilibrium integration procedure. The value of the **PEXIT** variable is computed as follows:

$$PEXIT = \frac{P_{exitstatic,desired}}{P_{ref}}$$

The variable P_{ref} is specified by the input variable **PREF**. Values of **PEXIT** <0.0 are not permitted. Naturally, poor convergence or solution divergence can occur if the value of **PEXIT** suggests boundary data which are significantly different from the remainder of the flowfield. In such cases where this occurs, it is recommended that the solution be started with more conservative boundary values, and then restarted using the final boundary values.

An alternate specification is provided for the **EXDT2DT** boundary specification as shown in the sample application above. In this case, three values are included following the original boundary specification line. The alternate specification is provided as a means of achieving a desired mass flow rate through the bounding surface using the **EXT2DT** algorithm. The desired mass flow rate is achieved iteratively by incrementally adjusting the exit static pressure specification until the desired flow rate is achieved. Therefore, in this specification, the variable **PEXIT** described in detail above is the initial exit static pressure used in the iterative process, **EMDOT** represents the desired mass flow rate through the bounding surface in pounds mass, and **PRELAX** is a relaxation factor to stabilize the iterative process (values may range from 0.0 to 1.0, though poor convergence is likely for values larger than 0.1). This procedure is not foolproof, and suffers from the fact that when a job is restarted, if an updated exit pressure is not inserted in the boundary data file, then the pressure-mass flow iterative process will essentially start over. The *ADPAC08* code will automatically determine when to employ the iterative process by identifying the additional boundary specification variables.

Restrictions/Limitations

The **EXT2DT** boundary condition assumes that the mesh is oriented in such a fashion that the radial coordinate is defined as $r = \sqrt{y^2 + z^2}$. For axial flow turbomachinery, this implies that the axis of rotation (or the centerline) coincides with the x axis. It is also required that the radial-like direction of the mesh be defined by the j coordinate, and is therefore not valid on a j =constant mesh plane. This is required in order to properly integrate the radial equilibrium equation to complete the exit static pressure specification. Examples of this type of mesh system can be found in the chapter defining standard configurations. The **EXT2DT** boundary specification is restricted to 2-D mesh surfaces (3-D mesh surfaces should use the **EXITT** boundary specification).

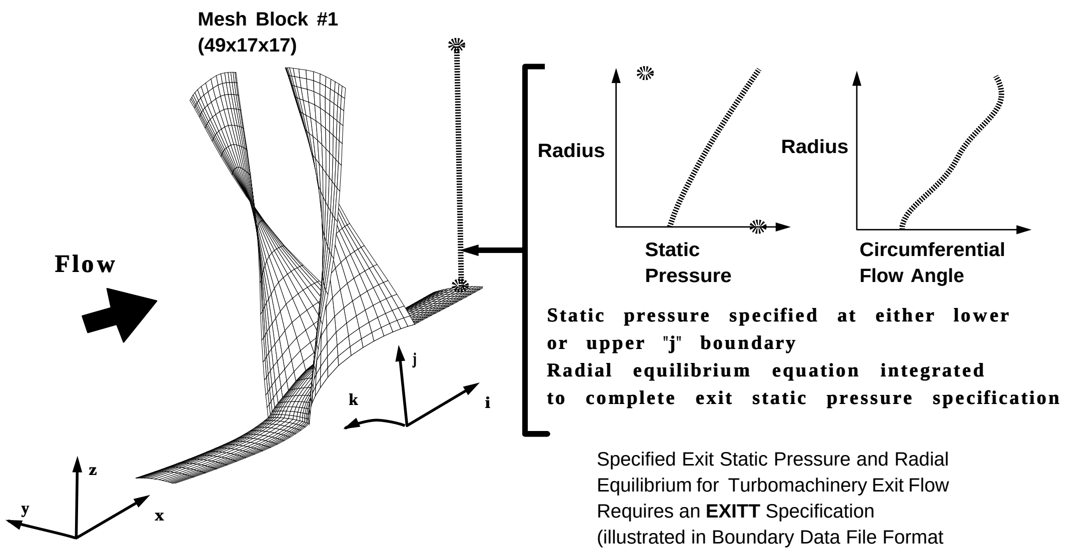
Common Errors

- Application of **EXT2DT** to a 3-D mesh system.

- Failure to specify the additional data value **PEXIT**.
- Improper specification of the alternate (mass flow) iterative scheme.
- Radial-like direction of the mesh is not the j coordinate.
- Cylindrical solution procedure not selected (**FCART**=1.0)
- Failure to properly specify the **LSPEC1** variable on the boundary data file specification line.
- Value of **PEXIT** is too high (flow cannot get started).

EXITT

Turbomachinery Exit Boundary Condition


Mesh Block #1
(49x17x17)
Flow
Radius
Static
Pressure
Radius
Circumferential
Flow Angle
Static pressure specified at either lower
or upper "j" boundary
Radial equilibrium equation integrated
to complete exit static pressure specification
j
k
i
z
y
x

Specified Exit Static Pressure and Radial Equilibrium for Turbomachinery Exit Flow Requires an **EXITT** Specification (illustrated in Boundary Data File Format statements below)

Application

The **EXITT** specification is used to impose a turbomachinery-based exit boundary condition based on radial equilibrium. The illustrative graphic above depicts an application of the **EXITT** outflow boundary condition for an H-type mesh for a turbomachinery fan rotor blade passage. The **EXITT** specification provides the radial variation of flow properties at the outflow boundary resulting from the application of a simplified form of the radial equilibrium equation. This boundary condition has been utilized extensively as an exit flow specifier for both ducted and unducted turbomachinery flows.

Boundary Data File Format

The boundary data file specification for the mesh surface indicated in the illustrative graphic for the **EXITT** boundary condition is given below:

```
EXITT 1  1  I  I  M  M  L L  49 49  1  17  1  17  1  17  1  17
PEXIT
1.105
```

or the alternate specification:

```
EXITT 1  1  I  I  M  M  L L  49 49  1  17  1  17  1  17  1  17
PEXIT    EMDOT    PRELAX
1.105    13.7     0.001
```

Note that a complete **EXITT** specification requires two additional lines following the **EXITT** boundary data file specification line. Failure to properly specify the data in these additional lines is a common **EXITT** specification error. It should also be mentioned that **EXITT** also requires proper specification of the **LSPEC1** variable for proper execution.

Description

The **EXITT** keyword specifies that a turbomachinery-based radial equilibrium exit flow boundary condition is to be applied to the mesh surface specified by **LFACE1** on the block specified by **LBLOCK1**. The **EXITT** boundary condition was specifically designed as an exit flow boundary procedure for axial and mixed flow turbomachinery geometries (pure radial flow turbomachinery exit flow boundaries may be usually be specified by the **EXITG** boundary condition). The **EXITT** boundary condition procedure utilizes a combination static pressure specification and integration of the radial equilibrium equation to define the static pressure field at all points on the boundary surface. As a result of the complexity of this procedure, several mesh restrictions were imposed to simplify the application of this approach. The primary assumption is that the integration of the radial equilibrium equation may be performed along the j coordinate direction of the mesh.

Hence, the j coordinate should be the radial-like direction. A single specification of static pressure is required at either the maximum or minimum extreme of the j coordinate of the boundary surface in order to initiate the integration process. The direction of integration and location of application of the specified exit static pressure are determined by the **LSPEC1** variable in the calling sequence. If **LSPEC1** = L, for LOW, then **PEXIT** is applied to the lower (smallest value) of the j index, and the radial equilibrium equation is integrated outward (increasing j direction). If **LSPEC1** = H, for HIGH, then **PEXIT** is applied to the upper (largest value) of the j index, and the radial equilibrium equation is integrated inward (decreasing j direction). The remaining flow variables on the **EXITT** boundary are updated by a Reimann invariant formulation based on the resulting local static pressure field. Included in the **EXITT** procedure is a special correction scheme which forces the flow to pass out of the flow domain. In other words, if the computed velocities result in a local inflow at the **EXITT** boundary, no matter how small the magnitude of the inflow, the velocities are reset to zero at that point. This boundary condition requires the specification of additional data, as shown in the boundary data format descriptor above. The first additional line following the **EXITT** specification is assumed to be a label and may contain any information; however, for consistency it is recommended that the label **PEXIT** be used. The line following the **PEXIT** label contains the value of specified non-dimensional exit static pressure used to initiate the radial equilibrium integration procedure. The value of the **PEXIT** variable is computed as follows:

$$PEXIT = \frac{P_{exitstatic,desired}}{P_{ref}}$$

The variable P_{ref} are specified by the input variable **PREF**. Values of **PEXIT** <0.0 are not permitted. Naturally, poor convergence or solution divergence can occur if the value of **PEXIT** suggests boundary data which are significantly different from the remainder of the flowfield. In such cases where this occurs, it is recommended that the solution be started with more conservative boundary values, and then restarted using the final boundary values.

An alternate specification is provided for the **EXITT** boundary specification as shown in the sample application above. In this case, three values are included following the original boundary specification line. The alternate specification is provided as a means of achieving a desired mass flow rate through the bounding surface using the **EXITT** algorithm. The desired mass flow rate is achieved iteratively by incrementally adjusting the exit static pressure specification until the desired flow rate is achieved. Therefore, in this specification, the variable **PEXIT** described in detail above is the initial exit static pressure used in the iterative process, **EMDOT** represents the desired mass flow rate through the bounding surface in pounds mass, and **PRELAX** is a relaxation factor to stabilize the iterative process (values may range from 0.0 to 1.0, though poor convergence is likely for values larger than 0.1). This procedure is not foolproof, and suffers from the fact that when a job is restarted, if an updated exit pressure is not inserted in the boundary data file, then the pressure-mass flow iterative process will essentially start over. The *ADPAC08* code will automatically determine when to employ the iterative process by identifying the additional boundary specification variables.

Restrictions/Limitations

The **EXITT** boundary condition assumes that the mesh is oriented in such a fashion that the radial coordinate is defined as $r = \sqrt{y^2 + z^2}$. For axial flow turbomachinery, this implies that the axis of rotation (or the centerline) coincides with the x axis. It is also required that the radial-like direction of the mesh be defined by the j coordinate, and is therefore not valid on a j =constant mesh plane. This is required in order to properly integrate the radial equilibrium equation to complete the exit static pressure specification. Examples of this type of mesh system can be found in the chapter defining standard configurations. The **EXITT** boundary specification is restricted to 3-D mesh surfaces (2-D mesh surfaces should use the **EXT2DT** boundary specification).

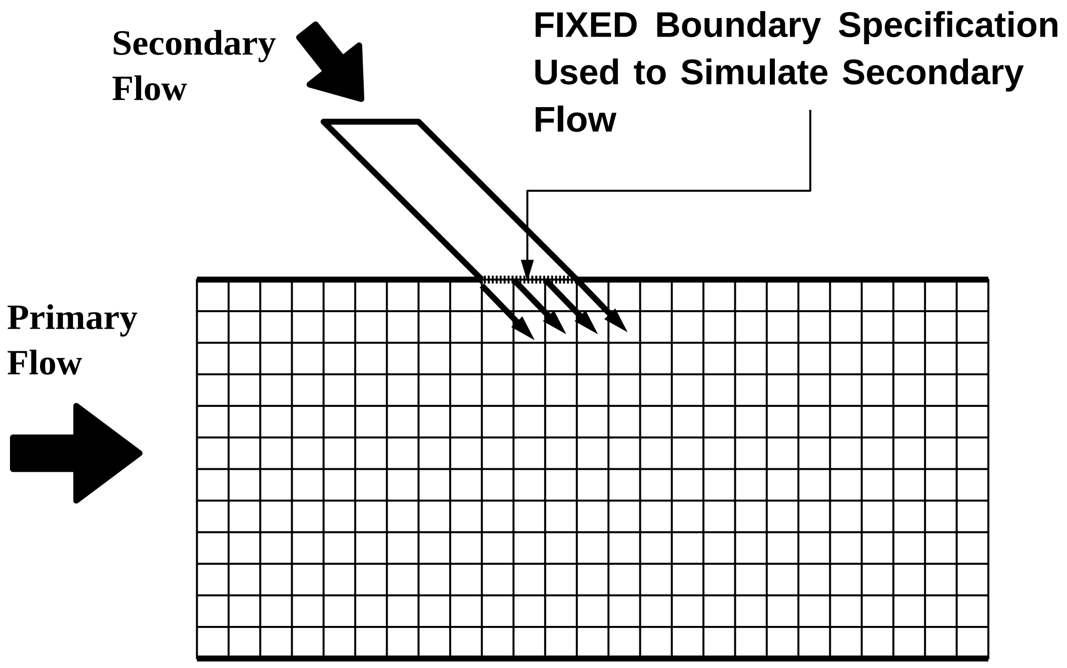
Common Errors

- Application of **EXITT** to a 2-D mesh system.
- Failure to specify the additional data value **PEXIT**.

- Improper specification of the alternate (mass flow) iterative scheme.
- Radial-like direction of the mesh is not the j coordinate.
- Mesh does not possess circumferential symmetry (axial, radial mesh coordinates vary in the circumferential coordinate direction).
- Failure to properly specify the **LSPEC1** variable on the boundary data file specification line.
- Value of **PEXIT** is too high (flow cannot get started).

FIXED

Fixed Flow Boundary Specification


Secondary Flow
FIXED Boundary Specification Used to Simulate Secondary Flow
Primary Flow

Application

The **FIXED** specification is used as a "last resort" boundary specification which hardwires flow properties into the numerical solution. The application illustrated above indicates an application of the **FIXED** boundary specification to provide a direct implementation of the flow properties of an injection jet into a simple duct flow. The same jet could have been modeled more effectively using alternate boundary conditions, or through the addition of an additional grid to simulate the jet flow passage; however, for the purposes of demonstration, and to obtain a solution of this type quickly, the **FIXED** specification was used instead.

Boundary Data File Format

The boundary data file specifications for the mesh interface indicated in the illustrative graphic for the **FIXED** boundary condition are given below:

```
FIXED 1 1 J J P P I K   1   1  11  21  1 11  11  21  1 11
RO     U      V      W    TTOT
0.002  100.0  100.0  0.0  600.0
```

Note that a complete **FIXED** specification requires the specification of additional data beyond the standard boundary specification line.

Description

The **FIXED** statement is used to provide a fixed specification of boundary flow data in the absence of any other appropriate boundary condition. This routine was provided for those cases where other boundary conditions either cannot provide the boundary specifications desired, or in those cases where a fixed boundary specification is deemed appropriate. In most cases, the **FIXED** specification is undesirable because the boundary condition itself is perfectly reflecting, and will therefore inhibit solution convergence. In addition, the **FIXED** specification does not permit interaction between the boundary flow and the interior flow, which runs contrary to the normal fluid dynamics behavior.

A **FIXED** specification requires two additional lines in addition to the normal boundary data file descriptor, as shown above. The first additional line simply contains the labels for the additional flow variable **RO**, **U**, **V**, **W**, and **TTOT**. The next line contains the actual values for the flow variable specifications. The variable **RO** defines the fluid density in slugs per cubic foot. The variables **U**, **V**, and **W** contain the fluid velocity components in feet per second for the x, y, and z coordinate directions for a Cartsian solution mesh block, and the x, r, and θ coordinate directions for a cylindrical solution mesh block, respectively. Finally, **TTOT** represents the fluid total temperature in degrees Rankine for the boundary specification. During the application of a **FIXED** specification, phantom boundary cell data are set according to the data provided in the extra lines following the boundary data specification line as shown above. As a result, the data is not necessarily applied *at* the boundary, but the influence of the data is felt just outside the boundary. This phenomenon is consistent with the behavior of a finite volume solution algorithm.

Restrictions/Limitations

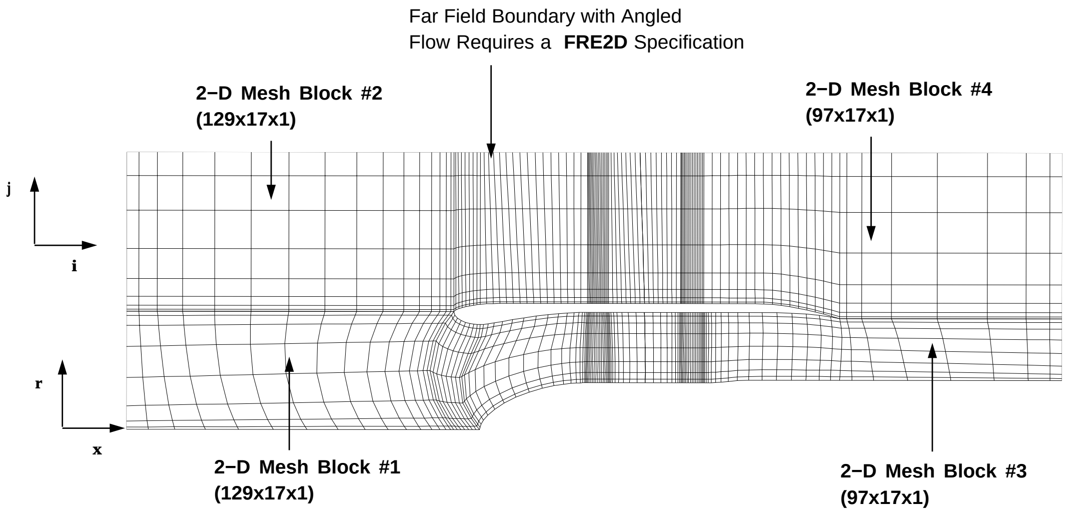
Data provided in the **FIXED** specification should represent phantom cell centered data and must be dimensionalized as described above.

Common Errors

- Incorrectly specified or misaligned extents of boundary regions (values of **M1LIM1, M1LIM2, N1LIM1, N1LIM2, M2LIM1, M2LIM2, N2LIM1, N2LIM2** do not correctly define the interface extents on blocks **LBLOCK1** and **LBLOCK2**).
- Failure to provide additional data for **FIXED** specification.
- **FIXED** boundary specifications for cylindrical solution mesh blocks must use the cylindrical velocity components.
- **FIXED** boundary specifications for Cartesian solution mesh blocks must use the Cartesian velocity components.

FRE2D

2-D Far Field Flow Boundary Condition



Application

The **FRE2D** specification is used to impose a far field boundary condition with uniform far field flow properties. The example graphic above illustrates a four block mesh system used to predict the axisymmetric flow through a high bypass ducted fan. The two outer blocks (#2 and #4) require a farfield boundary condition at the outer boundary (j=17). The **FRE2D** boundary specification is used to satisfy the farfield flow requirement. This boundary condition has been utilized extensively for both ducted and unducted 2-D fan propulsion systems.

Boundary Data File Format

The boundary data file specification for the mesh interface indicated in the illustrative graphic for the **FRE2D** boundary condition is given below:

```
FRE2D 2  2  J  J  M  M  I K 17  17  1 129  1   2  1 129  1   2  Block 2
PTOT   TTOT  EMINF   ALPHA
1.0    1.0    0.75    0.0
FRE2D 4  4  J  J  M  M  I K 17  17  1  97  1   2  1  97  1   2  Block 4
PTOT   TTOT  EMINF   ALPHA
1.0    1.0    0.75    0.0
```

Note that a complete **FRE2D** specification requires two additional lines following the **FRE2D** boundary data file specification line. Failure to properly specify the data in these additional lines is a common **FRE2D** specification error.

Description

The **FRE2D** statement specifies that an external, free flow boundary condition is to be applied to the mesh surface specified by **LFACE1** on the 2-D block specified by **LBLOCK1**. The **FRE2D** boundary condition is primarily used for external flow problems at a far field boundary to simulate the effects of the atmosphere or other large reservoir with known properties. The **FRE2D** procedure utilizes a Reimann invariant formulation to compute the local flow quantities, and permits both inflow and outflow through the bounding surface based on the nature of the local flow with respect to the known far field conditions. This boundary condition requires the specification of additional data, as shown in the boundary data format descriptor above. The first additional line following the **FRE2D** specification is assumed to be a label and may contain any information; however, for consistency it is recommended that the labels **PTOT**, **TTOT**, **EMINF**, and **ALPHA** be used. The next line contains the values imposed for the variables **PTOT**, **TTOT**, **EMINF**, and **ALPHA**, which represent the far field nondimensional reservoir total pressure and total temperature, along with the Mach number and Cartesian angle of attack, respectively, used in the **FRE2D** characteristic solution sequence. The value of the **PTOT** variable is the desired normalized far field total pressure computed as:

$$PTOT = \frac{P_{total,desired}}{P_{ref}}$$

and the value of the **TTOT** variable is the desired normalized far field total temperature computed as:

$$TTOT = \frac{T_{total,desired}}{T_{ref}}$$

The variables P_{ref} and T_{ref} are specified by the input variables **PREF** and **TREF**. Values of **PTOT** and **TTOT** $<$0.0 are not permitted. The variable **EMINF** represents the far field Mach number. The far field flow is *always* assumed to progress along the positive x axis, and therefore mesh systems should be generated with this in mind. Finally, the variable **ALPHA** represents the farfield Cartesian angle of attack, in degrees, relative to the x axis, with positive angles resulting in far field velocity components in the z coordinate direction. For 2-D flows, this must virtually always be zero. The angle of attack velocities are *always* in the x-z plane and the velocity components in the y coordinate direction are *always* zero. If there is outflow along the **FREE** boundary, then some small y component velocities may occur as a result of extrapolation from the near field flow. Naturally, poor convergence or solution divergence can occur if **PTOT**, **TTOT**, **EMINF** or **ALPHA** suggest boundary values which are significantly different from the remainder of the flowfield. In such cases where this occurs, it is recommended that the solution be started with more conservative boundary values, and then restarted using the final boundary values.

Restrictions/Limitations

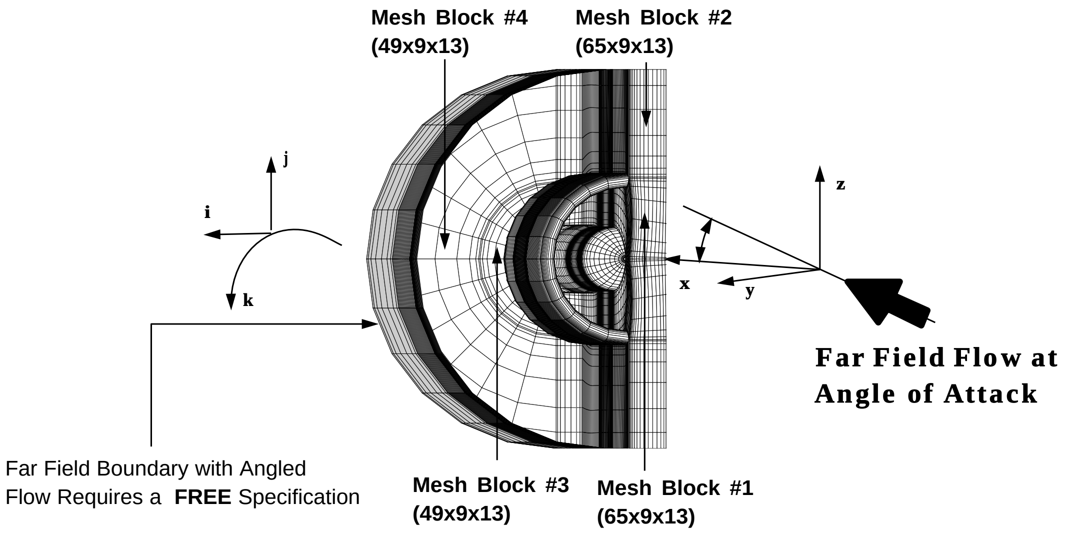
The **FRE2D** boundary specification is restricted to 2-D mesh surfaces (3-D mesh surfaces should use the **FREE** boundary specification). The far field flow angle of attack must usually be zero.

Common Errors

- Application of **FRE2D** to a 3-D mesh system.
- Failure to specify the additional data values **PTOT**, **TTOT**, **EMINF**, or **ALPHA**.
- **ALPHA** has a nonzero value.
- Failure to generate the mesh with $+x$ as the downstream flow direction.

FREE

Far Field Flow Boundary Condition



Application

The **FREE** specification is used to impose a far field boundary condition with uniform far field flow properties. The example graphic above illustrates a four block mesh system used to predict the 3-D flow through a high bypass ducted fan. The two outer blocks (#2 and #4) require a farfield boundary condition at the outer boundary (j=9). The **FREE** boundary specification is used to satisfy the farfield flow requirement This boundary condition has been utilized extensively for both ducted and unducted fan propulsion systems including angle of attack cases.

Boundary Data File Format

The boundary data file specification for the mesh interfaces indicated in the illustrative graphic for the **FREE** boundary condition are given below:

```
FREE 2  2  J  J  M  M  I K  9   9  1  65  1  13  1  65  1  13  Block 2
PTOT   TTOT  EMINF   ALPHA
1.0    1.0    0.75   10.0
FREE 4  4  J  J  M  M  I K  9   9  1  49  1  13  1  49  1  13  Block 4
PTOT   TTOT  EMINF   ALPHA
1.0    1.0    0.75   10.0
```

Note that a complete **FREE** specification requires two additional lines following the **FREE** boundary data file specification line. Failure to properly specify the data in these additional lines is a common **FREE** specification error.

Description

The **FREE** statement specifies that an external, free flow boundary condition is to be applied to the mesh surface specified by **LFACE1** on the block specified by **LBLOCK1**. The **FREE** boundary condition is primarily used for external flow problems at a far field boundary to simulate the effects of the atmosphere or other large reservoir with known properties. The **FREE** procedure utilizes a Reimann invariant formulation to compute the local flow quantities, and permits both inflow and outflow through the bounding surface based on the nature of the local flow with respect to the known far field conditions. This boundary condition requires the specification of additional data, as shown in the boundary data format descriptor above. The first additional line following the **FREE** specification is assumed to be a label and may contain any information; however, for consistency it is recommended that the labels **PTOT**, **TTOT**, **EMINF**, and **ALPHA** be used. The next line contains the values imposed for the variables **PTOT**, **TTOT**, **EMINF**, and **ALPHA**, which represent the far field nondimensional reservoir total pressure and total temperature, along with the Mach number and Cartesian angle of attack, respectively, used in the **FREE** characteristic solution sequence. The value of the **PTOT** variable is the desired normalized far field total pressure computed as:

$$PTOT = \frac{P_{total,desired}}{P_{ref}}$$

and the value of the **TTOT** variable is the desired normalized far field total temperature computed as:

$$TTOT = \frac{T_{total,desired}}{T_{ref}}$$

The variables P_{ref} and T_{ref} are specified by the input variables **PREF** and **TREF**. Values of **PTOT** and **TTOT** $<$0.0 are not permitted. The variable **EMINF** represents the far field Mach number. The far field flow is *always* assumed to progress along the positive x axis, and therefore mesh systems should be generated with this in mind. Finally, the variable **ALPHA** represents the farfield Cartesian angle of attack, in degrees, relative to the x axis, with positive angles resulting in far field velocity components in the z coordinate direction. The angle of attack velocities are *always* in the x-z plane and the velocity components in the y coordinate direction are *always* zero. If there is outflow along the **FREE** boundary, then some small y component velocities may occur as a result of extrapolation from the near field flow. Naturally, poor convergence or solution divergence can occur if **PTOT**, **TTOT**, **EMINF** or **ALPHA** suggest boundary values which are significantly different from the remainder of the flowfield. In such cases where this occurs, it is recommended that the solution be started with more conservative boundary values, and then restarted using the final boundary values.

Restrictions/Limitations

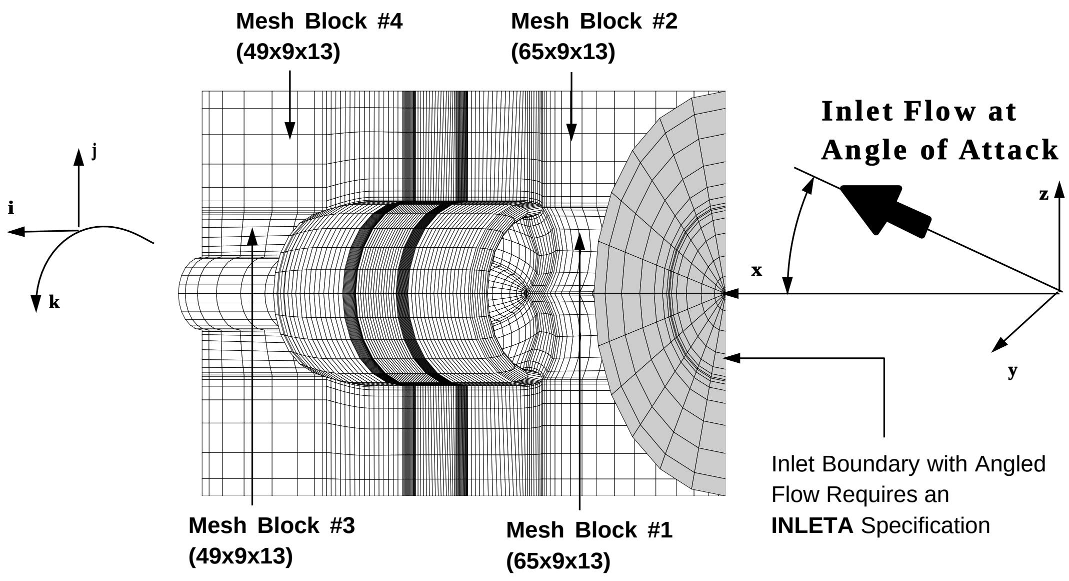
The **FREE** boundary specification is restricted to 3-D mesh surfaces (2-D mesh surfaces should use the **FRE2D** boundary specification). The far field flow angle of attack must be specified relative to the x axis, and produces additional velocity components in the z coordinate direction only. Imposed far-field velocity components in the y coordinate direction will always be zero.

Common Errors

- Application of **FREE** to a 2-D mesh system.
- Application of **FREE** to a boundary for which far field y coordinate direction velocity components are required.
- Failure to specify the additional data values **PTOT**, **TTOT**, **EMINF**, or **ALPHA**.
- Failure to generate the mesh with $+x$ as the downstream flow direction.

INLETA

Cartesian Angle of Attack Inflow Boundary Condition Procedure



Application

The **INLETA** specification is used to impose a Cartesian angle of attack inflow boundary condition with uniform flow properties at a local mesh surface. The illustrative graphic above depicts a four block mesh system for a turbofan engine geometry. The **INLETA** specifier is utilized at the inlet of mesh blocks 1 and 2 to set the angled inflow necessary to simulate angle of attack. This boundary condition has been utilized extensively as an inlet flow specifier for inlet, nacelle, and propfan geometries at angle of attack.

Boundary Data File Format

The boundary data file specification for the mesh boundaries indicated in the illustrative graphic for the **INLETA** boundary condition are given below:

```
INLETA 1  1  I  I  P  P  J K  1  1  1  9  1  13  1  9  1  13
PTOT   TTOT  ALPHA
1.0    1.0   20.0
INLETA 2  2  I  I  P  P  J K  1  1  1  9  1  13  1  9  1  13
PTOT   TTOT  ALPHA
1.0    1.0   20.0
```

Note that a complete **INLETA** specification requires two additional lines following the **INLETA** boundary data file specification line. Failure to properly specify the data in these additional lines is a common **INLETA** specification error.

Description

The **INLETA** keyword specifies that a uniform property angle of attack inflow boundary condition is to be applied to the mesh surface specified by **LFACE1** on the block specified by **LBLOCK1**. **INLETA** is valid for both cylindrical and Cartesian solution meshes (see the description of the input variable **FCART**). The **INLETA** procedure utilizes a Reimann invariant formulation to compute inflow velocities based on a specified upstream reservoir total pressure and total temperature, and a single Cartesian flow angle as shown in the illustrative graphic, above. Included in the **INLETA** procedure is a special correction scheme which forces the flow to pass into the flow domain. In other words, if the computed velocities result in a local outflow at the **INLETA** boundary, no matter how small the magnitude of the outflow, the velocities are reset to zero at that point. This boundary condition requires the specification of additional data, as shown in the boundary data format descriptor above. The first additional line following the **INLETA** specification is assumed to be a label and may contain any information; however, for consistency it is recommended that the labels **PTOT**, **TTOT**, and **ALPHA** be used. The next line contains the values imposed for the variables

PTOT, **TTOT** and **ALPHA** which represent the upstream reservoir total pressure, total temperature, and Cartesian flow angle, respectively, used in the **INLETA** characteristic solution sequence. The value of the **PTOT** variable is the desired normalized upstream total pressure computed as:

$$PTOT = \frac{P_{total,desired}}{P_{ref}}$$

and the value of the **TTOT** variable is the desired normalized upstream total temperature computed as:

$$TTOT = \frac{T_{total,desired}}{T_{ref}}$$

The variables P_{ref} and T_{ref} are specified by the input variables **PREF** and **TREF**. Values of **PTOT** and **TTOT** <0.0 are not permitted. The variable **ALPHA** represents the flow angle in degrees referenced to the x axis. Positive flow angles generate components of the flow in the positive z direction. In any case, the inlet velocity component in the y direction is set to zero. Values of **ALPHA** must lie between +/- 90 degrees. Naturally, poor convergence or solution divergence can occur if **PTOT** or **TTOT** suggest boundary values which are significantly different from the remainder of the flowfield, or if **ALPHA** is very large. In such cases where this occurs, it is recommended that the solution be started with more conservative boundary values, and then restarted using the final boundary values.

Restrictions/Limitations

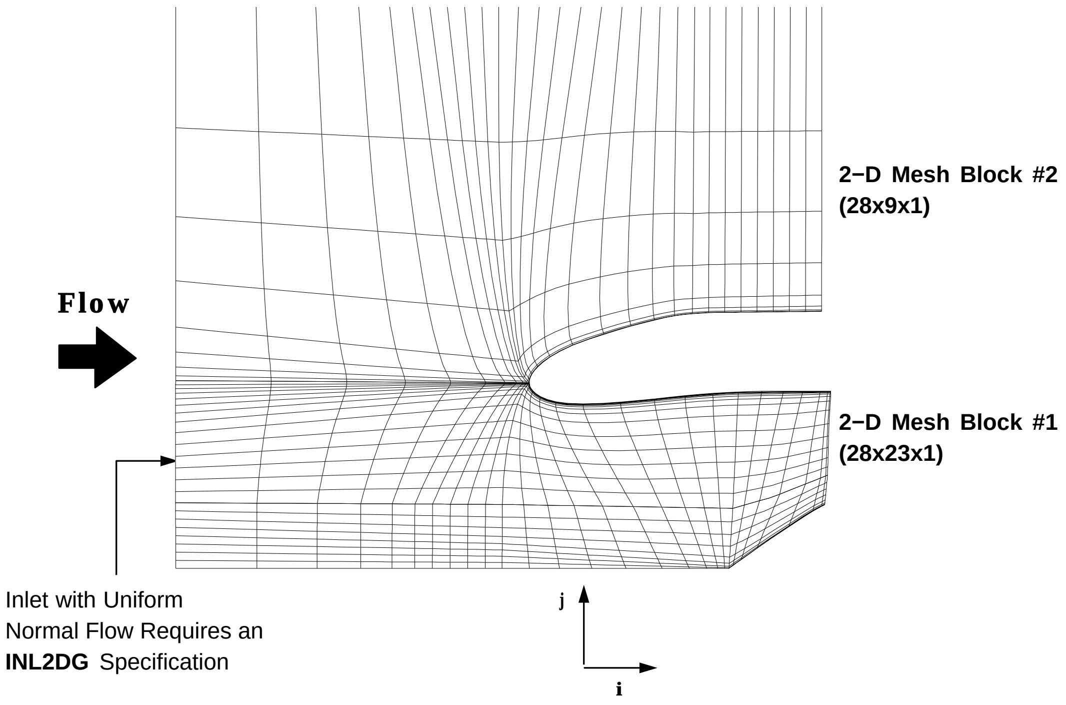
The **INLETA** boundary specification is restricted to 3-D mesh surfaces (There is no equivalent 2-D mesh surface related to angle of attack. Additional details are available under the description of the **INL2DA** boundary specification). The angle of the velocity components specified by the **INLETA** procedure must always be referenced to the x coordinate axis, and it is left to the user to generate a mesh which is consistent with this feature.

Common Errors

- Application of **INLETA** to a 2-D mesh system.
- Application of **INLETA** to a boundary for which non-zero y component velocities are required.
- Failure to specify the additional data values **PTOT**, **TTOT**, or **ALPHA**.

INL2DG

Generic Inflow Boundary Condition



Application

The **INL2DG** specification is used to impose a generic inflow boundary condition with uniform flow properties where the inflow velocity is normal to the local mesh surface. The example graphic above illustrates a 2-D 2-block mesh system mixing two adjacent streams of varying inlet properties. In this case, the **INL2DG** boundary specification is used to set the inflow boundary separately for each block to provide the desired incoming stream flow properties. This boundary condition has been utilized extensively as an inlet flow specifier for 2-D duct flows.

Boundary Data File Format

The boundary data file specification for the two mesh surfaces indicated in the illustrative graphic for the **INL2DG** boundary condition are given below:

```
INL2DG 1  1  I  I  P  P  J K  1   1  1  23  1   2  1  23  1   2
PTOT   TTOT
1.0    1.0
INL2DG 2  2  I  I  P  P  J K  1   1  1   9  1   2  1   9  1   2
PTOT   TTOT
1.2    1.8
```

Note that a complete **INL2DG** specification requires two additional lines following the **INL2DG** boundary data file specification line. Failure to properly specify the data in these additional lines is a common **INL2DG** specification error.

Description

The **INL2DG** statement specifies that a generic, uniform normal inflow boundary condition is to be applied to the 2-D mesh surface specified by **LFACE1** on the block specified by **LBLOCK1**. The **INL2DG** boundary condition should be applied for those cases where any other "specialized" inflow boundary condition (such as **INL2DT**, etc.) does not apply. The **INL2DG** boundary condition is also likely to be somewhat more efficient computationally than the other inflow boundary condition procedures, at the expense of some physical simplification. **INL2DG** is valid for either cylindrical or Cartesian-based (see the input variable **FCART**) solutions on 2-D meshes. The **INL2DG** procedure utilizes a Reimann invariant formulation to compute inflow velocities based on a specified upstream reservoir total pressure and total temperature. The velocity components at an **INL2DG** boundary are always computed to be normal (no transverse velocity components) to the local cell face at which the procedure is applied. Included in the **INL2DG** procedure is a special correction scheme which forces the flow to pass into the flow domain. In other words, if the computed velocities result in a local outflow at the **INL2DG** boundary, no matter how small the magnitude of the outflow, the velocities are reset to zero at that point. This boundary condition requires the specification of additional data, as shown in the boundary data format descriptor above. The first additional line following the **INL2DG** specification is assumed to be a label and may contain any information; however, for consistency it is recommended that the labels **PTOT** and **TTOT** be used. The next line contains the values imposed for the variables **PTOT** and **TTOT**, which represent the upstream reservoir total pressure and total temperature, respectively, used in the **INL2DG** characteristic solution sequence. The value of the **PTOT** variable is the desired normalized upstream total pressure computed as:

$$PTOT = \frac{P_{total,desired}}{P_{ref}}$$

and the value of the **TTOT** variable is the desired normalized upstream total temperature computed as:

$$TTOT = \frac{T_{total,desired}}{T_{ref}}$$

The variables P_{ref} and T_{ref} are specified by the input variables **PREF** and **TREF**. Values of **PTOT** and **TTOT** <0.0 are not permitted. Naturally, poor convergence or solution divergence can occur if **PTOT** or **TTOT** suggest boundary values which are significantly different from the remainder of the flowfield. In such cases where this occurs, it is recommended that the solution be started with more conservative boundary values, and then restarted using the final boundary values.

Restrictions/Limitations

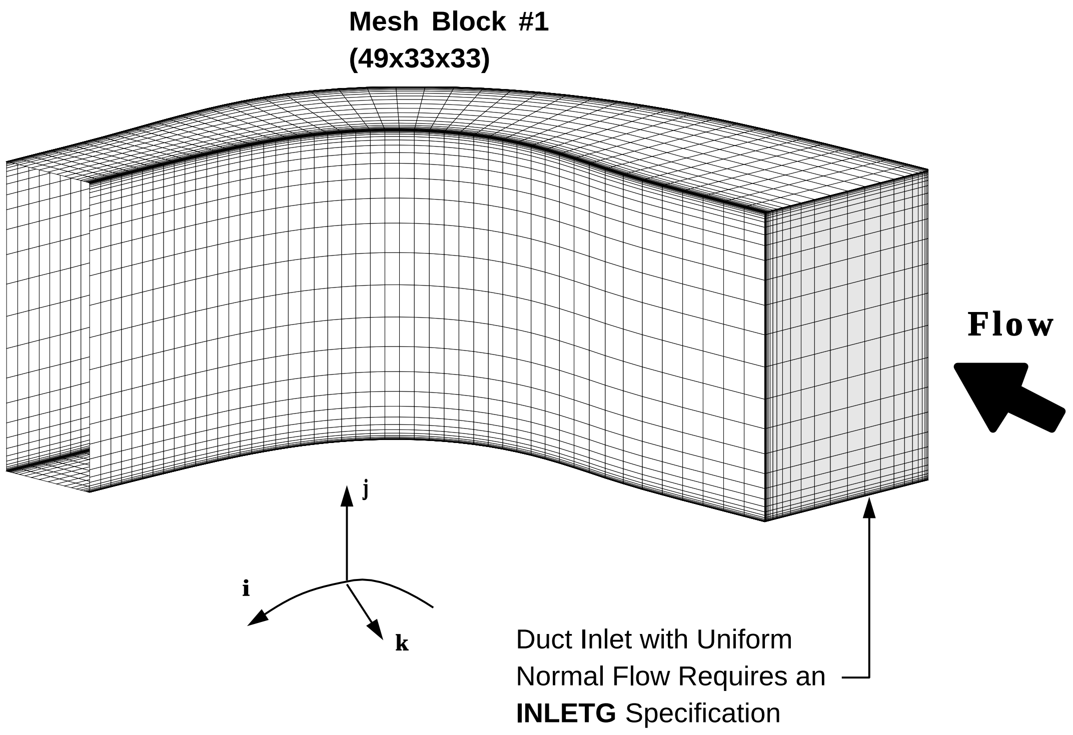
The **INL2DG** boundary specification is restricted to 2-D mesh surfaces (3-D mesh surfaces should use the **INLETG** boundary specification).

Common Errors

- Application of **INL2DG** to a 3-D mesh system.
- Application of **INL2DG** to a boundary for which transverse inflow velocity components are required.
- Failure to specify the additional data values **PTOT** or **TTOT**.

INLETG

Generic Inflow Boundary Condition



Application

The **INLETG** specification is used to impose a generic inflow boundary condition with uniform flow properties where the inflow velocity is normal to the local mesh surface. This boundary condition has been utilized extensively as an inlet flow specifier for for duct flows and turbine blade cooling flow.

Boundary Data File Format

The boundary data file specification for the mesh interface indicated in the illustrative graphic for the **INLETG** boundary condition is given below:

```
INLETG 1  1  I  I  P  P  J K  1   1  1  33  1  33  1  33  1  33
PTOT   TTOT
1.0    1.0
```

Note that a complete **INLETG** specification requires two additional lines following the **INLETG** boundary data file specification line. Failure to properly specify the data in these additional lines is a common **INLETG** specification error.

Description

The **INLETG** statement specifies that a generic, uniform normal inflow boundary condition is to be applied to the mesh surface specified by **LFACE1** on the block specified by **LBLOCK1**. The **INLETG** boundary condition should be applied for those cases where any other "specialized" inflow boundary condition (such as **INLETR, INLETT**, etc.) does not apply. The **INLETG** boundary condition is also likely to be somewhat more efficient computationally than the other inflow boundary condition procedures, at the expense of some physical simplification. **INLETG** may be utilized on either cylindrical or Cartesian solution meshes (see the description of the input variable **FCART**). The **INLETG** procedure utilizes a Reimann invariant formulation to compute inflow velocities based on a specified upstream reservoir total pressure and total temperature. The velocity components at an **INLETG** boundary are always computed to be normal (no transverse velocity components) to the local cell face at which the procedure is applied. Included in the **INLETG** procedure is a special correction scheme which forces the flow to pass into the flow domain. In other words, if the computed velocities result in a local outflow at the **INLETG** boundary, no matter how small the magnitude of the outflow, the velocities are reset to zero at that point. This boundary condition requires the specification of additional data, as shown in the boundary data format descriptor above. The first additional line following the **INLETG** specification is assumed to be a label and may contain any information; however, for consistency it is recommended that the labels **PTOT** and **TTOT** be used. The next line contains the values imposed for the variables **PTOT** and **TTOT**, which represent the upstream reservoir total pressure and total temperature, respectively, used in the **INLETG** characteristic solution sequence. The value of the **PTOT** variable is the desired normalized upstream total pressure computed as:

$$PTOT = \frac{P_{total,desired}}{P_{ref}}$$

and the value of the **TTOT** variable is the desired normalized upstream total temperature computed as:

$$TTOT = \frac{T_{total,desired}}{T_{ref}}$$

The variables P_{ref} and T_{ref} are specified by the input variables **PREF** and **TREF**. Values of **PTOT** and **TTOT** <0.0 are not permitted. Naturally, poor convergence or solution divergence can occur if **PTOT** or **TTOT** suggest boundary values which are significantly different from the remainder of the flowfield. In such cases where this occurs, it is recommended that the solution be started with more conservative boundary values, and then restarted using the final boundary values.

Restrictions/Limitations

The **INLETG** boundary specification is not restricted to 3-D mesh surfaces (although for consistency 2-D mesh surfaces should use the **INL2DG** boundary specification).

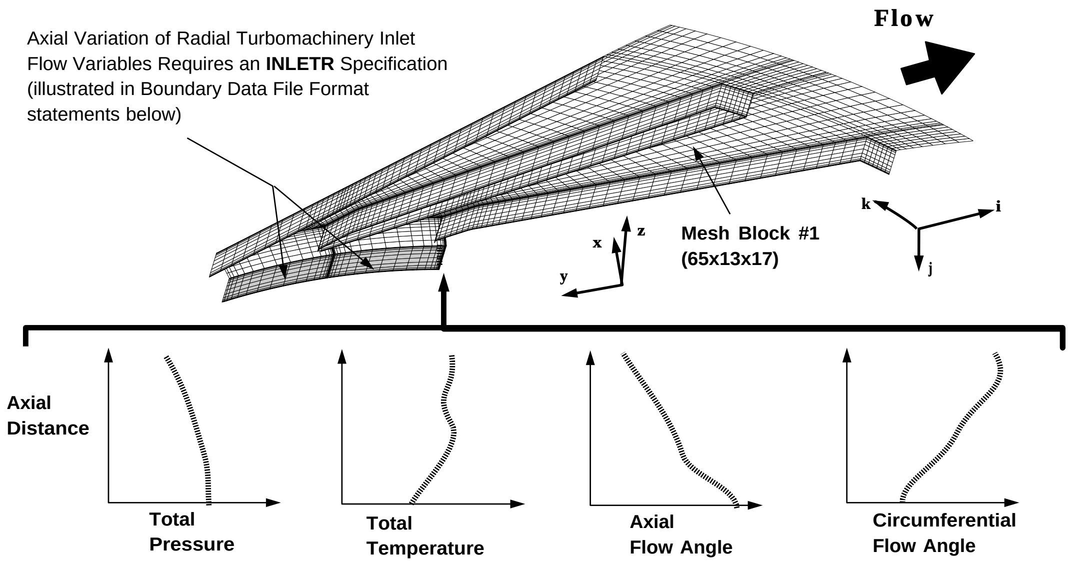
Common Errors

- Application of **INLETG** to a boundary for which transverse inflow velocity components are required.

- Failure to specify the additional data values **PTOT** or **TTOT**.

INLETR

Radial Flow Turbomachinery Inflow Boundary Condition


Axial Variation of Radial Turbomachinery Inlet Flow Variables Requires an **INLETR** Specification (illustrated in Boundary Data File Format statements below)
Flow
Mesh Block #1 (65x13x17)
x
y
z
k
i
j
Axial Distance
Total Pressure
Total Temperature
Axial Flow Angle
Circumferential Flow Angle

Application

The **INLETR** specification is used to impose an inflow boundary condition with axially varying flow properties for radial flow turbomachinery. The example graphic above illustrates adjacent passages of a mesh system designed to predict the flow through a radial diffuser. The inlet boundary is a radial surface of revolution with properties which vary in the *axial* direction, and therefore **INLETR** is used to supply the desired flow characteristics at this boundary. **INLETR** has been successfully used for several radial flow turbomachinery geometries.

Boundary Data File Format

The boundary data file specification for the mesh surface indicated in the illustrative graphic for the **INLETR** boundary condition is given below:

```
INLETR 1  1  I  I  P  P  J K  1   1  1  13  1  17  1  13  1  17
NDATA
 4
AXIAL  PTOT   TTOT   BETAX   BETAT
0.1    0.99   0.99   5.0     -73.3
0.2    0.98   1.01   4.0     -75.8
0.3    0.97   1.00   3.0     -77.2
0.4    0.96   1.01   2.0     -79.0
```

Note that a complete **INLETR** specification requires at least six additional lines following the **INLETR** boundary data file specification line. Failure to properly specify the data in these additional lines is a common **INLETR** specification error.

Description

The **INLETR** statement specifies that an axial flow turbomachinery inlet flow boundary condition with axially varying flow properties is to be applied to the mesh surface specified by **LFACE1** on the block specified by **LBLOCK1**. The **INLETR** boundary condition was specifically designed as an inflow boundary procedure for pure radial flow turbomachinery geometries (axial and mixed flow turbomachinery inflow boundaries may be specified by the **INLETT** boundary condition). The **INLETR** procedure utilizes a Reimann invariant formulation to compute inflow velocities based on a specified axial variation in flow properties (upstream reservoir total pressure,total temperature, axial flow angle, and circumferential flow angle). Included in the **INLETR** procedure is a special correction scheme which forces the flow to pass into the flow domain. In other words, if the computed velocities result in a local outflow at the **INLETR** boundary, no matter how small the magnitude of the outflow, the velocities are reset to zero at that point. This boundary condition requires the specification of additional data, as shown in the boundary data format descriptor above. The first additional line following the **INLETR** specification is assumed to be a label and may contain any information; however, for consistency it is recommended that the label **NDATA** be used. The line following the **NDATA** label contains the number of axial data points which will be used to specify the desired axial variation of properties at the inflow boundary. At least 3 axial data locations must be specified to use the **INLETR** boundary condition. The third line following the **INLETR** specifier is again a label which outlines the variables **AXIAL, PTOT, TTOT, BETAX** and **BETAT**. The remaining **NDATA** lines contain the numeric information which defines the axial variation of the flow properties specified by these variables. The variable **AXIAL** is the axial coordinate (remember, the centerline is the x axis) at which the data is specified. This value should be nondimensionalized in the same manner as the mesh is nondimensionalized. This implies that the **AXIAL** variable, when multiplied by the input variable **DIAM** will result in the true geometric measurement in feet. Due to the interpolation procedures which will ultimately be performed on the **NDATA** lines of radial inflow data, it is essential that the axial variations be specified in a monotonic (constantly increasing) fashion. The variables **PTOT** and **TTOT** represent the local upstream reservoir total pressure and total temperature, respectively, used in the **INLETR** characteristic solution sequence. The value of the **PTOT** variable is the desired normalized upstream total pressure computed as:

$$PTOT = \frac{P_{total,desired}}{P_{ref}}$$

and the value of the **TTOT** variable is the desired normalized upstream total temperature computed as:

$$TTOT = \frac{T_{total,desired}}{T_{ref}}$$

The variables P_{ref} and T_{ref} are specified by the input variables **PREF** and **TREF**. Values of **PTOT** and **TTOT** <0.0 are not permitted. The variables **BETAX** and **BETAT** represent the local axial and circumferential flow angles expressed in degrees according to the coordinate orientation defined in Figure 3.7.

Naturally, poor convergence or solution divergence can occur if any of

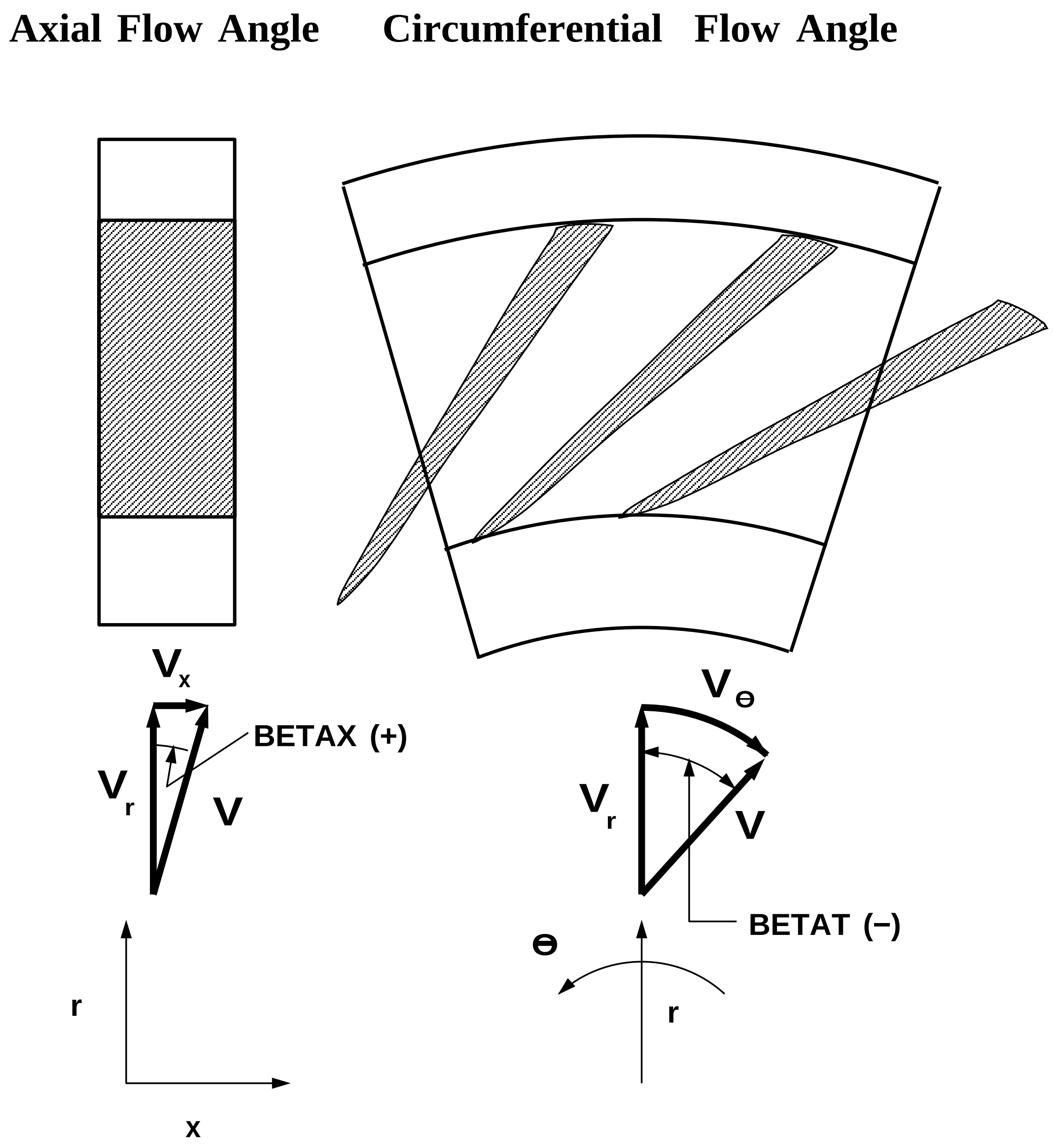


Figure 3.7: *ADPAC08* **INLETR** Boundary Specification Flow Angle Reference

the values of **PTOT**,**TTOT**,**BETAX**, or **BETAT** suggest boundary values which are significantly different from the remainder of the flowfield, or if the axial variation of these values is excessively large. In such cases where this occurs, it is recommended that the solution be started with more conservative boundary values, and then restarted using the final boundary values.

Restrictions/Limitations

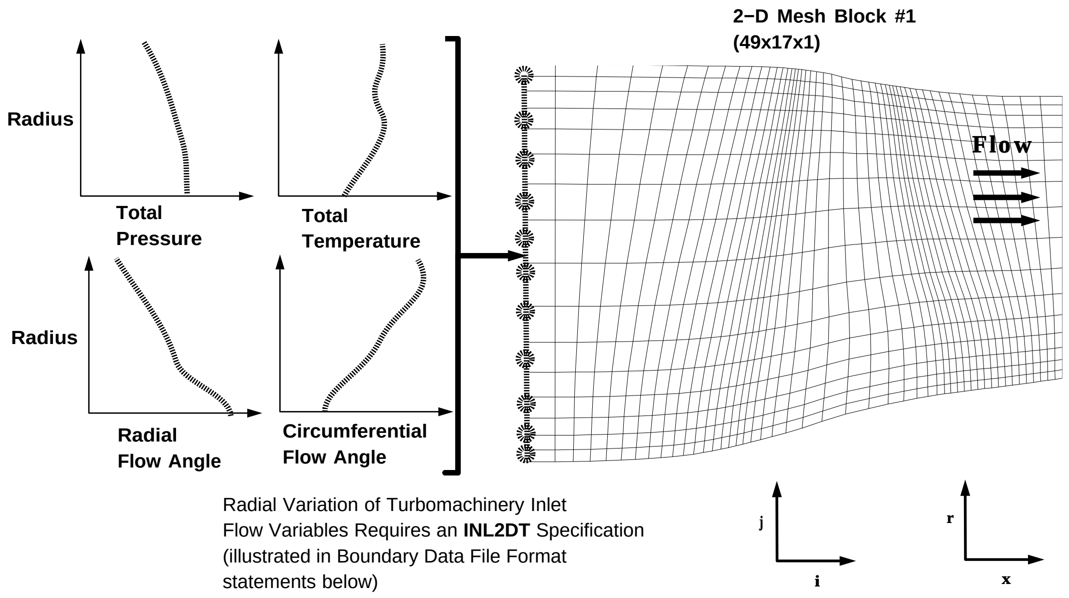
The **INLETR** boundary condition assumes that the mesh is oriented in such a fashion that the radial coordinate is defined as $r = \sqrt{y^2 + z^2}$. For radial flow turbomachinery, this implies that the axis of rotation (or the centerline) coincides with the x axis. It is also required that the axial-like direction of the mesh be defined by the j coordinate. An example of this type of mesh system can be found in the illustrative graphic included at the beginning of this description. The **INLETR** boundary specification is restricted to 3-D mesh surfaces.

Common Errors

- Application of **INLETR** to a 2-D mesh system.
- Failure to specify the additional data values **NDATA**, **AXIAL**, **PTOT**, **TTOT**, **BETAX**. or **BETAT**.
- Axial-like direction of the mesh is not the j coordinate.
- **NDATA** less than 3, resulting in improper interpolation.
- **BETAX** and/or **BETAT** orientation incorrectly interpreted.
- **AXIAL**, **PTOT** and/or **TTOT** improperly normalized.
- Mesh/geometry not defined with the x axis as the centerline.

INL2DT

2-D Turbomachinery Inflow Boundary Condition



Application

The **INL2DT** specification is used to impose an inflow boundary condition with radially varying flow properties for 2-D axisymmetric mesh systems. The example graphic illustrated above depicts an **EXT2DT** specification for a 2-D (axisymmetric) flow solution for a turbomachinery blade row. This boundary condition has been utilized extensively as an inlet flow specifier for 2-D turbomachinery flow passages and solutions for embedded blade rows with imposed axisymmetric body forces.

Boundary Data File Format

The boundary data file specification for the mesh surface indicated in the illustrative graphic for the **INL2DT** boundary condition is given below:

```
INL2DT 1  1  I  I  P  P  J K  1   1  1  17  1   2  1  17  1   2
NDATA
 7
RAD  PTOT   TTOT   BETAR   BETAT
0.20 1.01   0.98   5.0      5.1
0.25 1.01   0.99   4.0      5.7
0.30 1.00   1.00   3.0      6.3
0.35 0.99   1.01   2.5      6.8
0.40 0.97   1.00   2.0      7.4
0.45 0.96   1.01   1.0      8.0
0.50 0.95   1.01   0.0      7.7
```

Note that a complete **INL2DT** specification requires at least six additional lines following the **INL2DT** boundary data file specification line. Failure to properly specify the data in these additional lines is a common **INL2DT** specification error.

Description

The **INL2DT** statement specifies that a turbomachinery-based radially varying inflow boundary condition is to be applied to the 2-D mesh surface specified by **LFACE1** on the block specified by **LBLOCK1**. The **INL2DT** boundary condition was specifically designed as an inflow boundary procedure for axial and mixed flow axisymmetric turbomachinery geometries (radial flow turbomachinery inflow boundaries may be specified by the **INL2DR** boundary condition). The **INL2DT** procedure utilizes a Reimann invariant formulation to compute inflow velocities based on a specified radial variation in flow properties (upstream reservoir total pressure,total temperature, radial flow angle, and circumferential flow angle). Included in the **INL2DT** procedure is a special correction scheme which forces the flow to pass into the flow domain. In other words, if the computed velocities result in a local outflow at the **INL2DT** boundary, no matter how small the magnitude of the outflow, the velocities are reset to zero at that point. This boundary condition requires the specification of additional data, as shown in the boundary data format descriptor above. The first additional line following the **INL2DT** specification is assumed to be a label and may contain any information; however, for consistency it is recommended that the label **NDATA** be used. The line following the **NDATA** label contains the number of radial data points which will be used to specify the desired radial variation of properties at the inflow boundary. At least 3 radial data locations must be specified to use the **INL2DT** boundary condition. The third line following the **INL2DT** specifier is again a label which outlines the variables **RAD, PTOT, TTOT, BETAR** and **BETAT**. The remaining **NDATA** lines contain the numeric information which defines the radial variation of the flow properties specified by these variables. The variable **RAD** is the radius (remember, the centerline is the x axis) at which the data is specified. This value should be nondimensionalized in the same manner as the mesh is nondimensionalized. This implies that the **RAD** variable, when multiplied by the input variable **DIAM** will result in the true geometric measurement in feet. Due to the interpolation procedures which will ultimately be performed on the **NDATA** lines of radial inflow data, it is essential that the radial variations be specified from the inner to the outer radius in a monotonic (constantly increasing) fashion. The variables **PTOT** and **TTOT** represent the local upstream reservoir total pressure and total temperature, respectively, used in the **INL2DT** characteristic solution sequence. The value of the **PTOT** variable is the desired normalized upstream total pressure computed as:

$$PTOT = \frac{P_{total,desired}}{P_{ref}}$$

and the value of the **TTOT** variable is the desired normalized upstream total temperature computed as:

$$TTOT = \frac{T_{total,desired}}{T_{ref}}$$

The variables P_{ref} and T_{ref} are specified by the input variables **PREF** and **TREF**. Values of **PTOT** and **TTOT** <0.0 are not permitted. The variables **BETAR** and **BETAT** represent the local radial and circumferential flow angles expressed in degrees according to the coordinate orientation defined in Figure 3.8. Naturally, poor convergence or solution divergence can occur if any of the values of **PTOT**,**TTOT**,**BETAR**, or **BETAT** suggest boundary values which are significantly different from the remainder of the flowfield, or if the radial variation of these values is excessively large. In such cases where this occurs, it is recommended that the solution be started with more conservative boundary values, and then restarted using the final boundary values.

Restrictions/Limitations

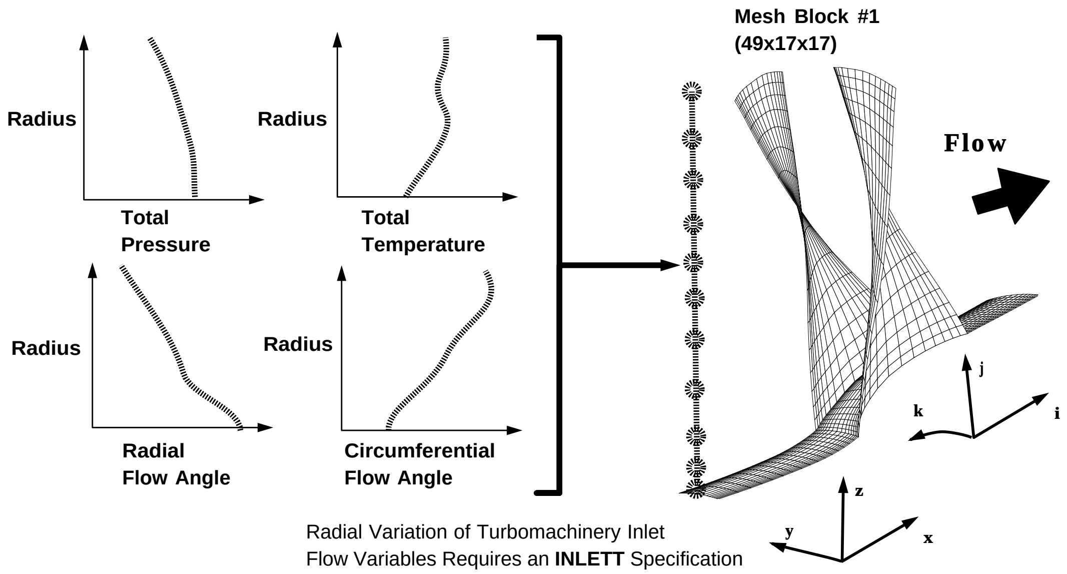
The **INL2DT** boundary condition assumes that the mesh is oriented in such a fashion that the radial coordinate is defined as $r = \sqrt{y^2 + z^2}$. For axial flow turbomachinery, this implies that the axis of rotation (or the centerline) coincides with the x axis. It is also required that the radial-like direction of the mesh be defined by the j coordinate. Examples of this type of mesh system can be found in the chapter defining standard configurations. The **INL2DT** boundary specification is restricted to 2-D mesh surfaces (3-D mesh surfaces should use the **INLETT** boundary specification).

Common Errors

- Application of **INL2DT** to a 3-D mesh system.
- Failure to specify the additional data values **NDATA**, **PTOT**, **TTOT**, **BETAR**. or **BETAT**.
- Radial-like direction of the mesh is not the j coordinate.
- **NDATA** less than 3, resulting in improper interpolation.
- **BETAR** and/or **BETAT** orientation incorrectly interpreted.
- **RAD**, **PTOT** and/or **TTOT** improperly normalized.
- Mesh/geometry not defined with the x axis as the centerline.

INLETT

Turbomachinery Inflow Boundary Condition



Radial Variation of Turbomachinery Inlet Flow Variables Requires an **INLETT** Specification (illustrated in Boundary Data File Format statements below)

Application

The **INLETT** specification is used to impose an inflow boundary condition with radially varying flow properties. The illustrative graphic above depicts an application of the **INLETT** inflow boundary condition for an H-type mesh for a turbomachinery fan rotor blade passage. The **INLETT** specification provides the radial variation of flow properties at the inflow boundary resulting from experimental conditions, upstream blade rows, or other known inlet property variation. This boundary condition has been utilized extensively as an inlet flow specifier for turbomachinery blade passages and annular ducts.

Boundary Data File Format

The boundary data file specification for the mesh surface indicated in the illustrative graphic for the **INLETT** boundary condition is given below:

```
INLETT 1  1  I  I  P  P  J K  1   1  1  17  1  17  1  17  1  17
NDATA
 7
RAD  PTOT   TTOT   BETAR   BETAT
0.20 1.01   0.98   5.0      5.1
0.25 1.01   0.99   4.0      5.7
0.30 1.00   1.00   3.0      6.3
0.35 0.99   1.01   2.5      6.8
0.40 0.97   1.00   2.0      7.4
0.45 0.96   1.01   1.0      8.0
0.50 0.95   1.01   0.0      7.7
```

Note that a complete **INLETT** specification requires six or more additional lines following the **INLETT** boundary data file specification line. Failure to properly specify the data in these additional lines is a common **INLETT** specification error.

Description

The **INLETT** statement specifies that a turbomachinery-based radially varying inflow boundary condition is to be applied to the mesh surface specified by **LFACE1** on the block specified by **LBLOCK1**. The **INLETT** boundary condition was specifically designed as an inflow boundary procedure for axial and mixed flow turbomachinery geometries (radial flow turbomachinery inflow boundaries may be specified by the **INLETR** boundary condition). As such, the **INLETT** boundary procedure is only valid on mesh systems employing the cylindrical solution algorithm (see the description of the input variable **FCART**). The **INLETT** procedure utilizes a Reimann invariant formulation to compute inflow velocities based on a specified radial variation in flow properties (upstream reservoir total pressure,total temperature, radial flow angle, and circumferential flow angle). Included in the **INLETT** procedure is a special correction scheme which forces the flow to pass into the flow domain. In other words, if the computed velocities result in a local outflow at the **INLETT** boundary, no matter how small the magnitude of the outflow, the velocities are reset to zero at that point. This boundary condition requires the specification of additional data, as shown in the boundary data format descriptor above. The first additional line following the **INLETT** specification is assumed to be a label and may contain any information; however, for consistency it is recommended that the label **NDATA** be used. The line following the **NDATA** label contains the number of radial data points which will be used to specify the desired radial variation of properties at the inflow boundary. At least 3 radial data locations must be specified to use the **INLETT** boundary condition. The third line following the **INLETT** specifier is again a label which outlines the variables **RAD, PTOT, TTOT, BETAR** and **BETAT**. The remaining **NDATA** lines contain the numeric information which defines the radial variation of the flow properties specified by these variables. The variable **RAD** is the radius (remember, the centerline is the x axis) at which the data is specified. This value should be nondimensionalized in the same manner as the mesh is nondimensionalized. This implies that the **RAD** variable, when multiplied by the input variable **DIAM** will result in the true geometric measurement in feet. Due to the interpolation procedures which will ultimately be performed on the **NDATA** lines of radial inflow data, it is essential that the radial variations be specified from the inner to the outer radius in a monotonic (constantly increasing) fashion. The variables **PTOT** and **TTOT** represent the local upstream reservoir total pressure and total temperature, respectively, used in the **INLETT** characteristic solution sequence. The value of the **PTOT** variable is the desired normalized upstream total pressure computed as:

$$PTOT = \frac{P_{total,desired}}{P_{ref}}$$

and the value of the **TTOT** variable is the desired normalized upstream total temperature computed as:

$$TTOT = \frac{T_{total,desired}}{T_{ref}}$$

The variables P_{ref} and T_{ref} are specified by the input variables **PREF** and **TREF**. Values of **PTOT** and **TTOT** <0.0 are not permitted. The variables

BETAR and **BETAT** represent the local radial and circumferential flow angles expressed in degrees according to the coordinate orientation defined in Figure 3.8. Naturally, poor convergence or solution divergence can occur if any of the values of **PTOT**,**TTOT**,**BETAR**, or **BETAT** suggest boundary values which are significantly different from the remainder of the flowfield, or if the radial variation of these values is excessively large. In such cases where this occurs, it is recommended that the solution be started with more conservative boundary values, and then restarted using the final boundary values.

Restrictions/Limitations

The **INLETT** boundary condition assumes that the mesh is oriented in such a fashion that the radial coordinate is defined as $r = \sqrt{y^2 + z^2}$. For axial flow turbomachinery, this implies that the axis of rotation (or the centerline) coincides with the x axis. It is also required that the radial-like direction of the mesh be defined by the j coordinate. Examples of this type of mesh system can be found in the chapter defining standard mesh configurations. The **INLETT** boundary specification is restricted to 3-D mesh surfaces (2-D mesh surfaces should use the **INL2DT** boundary specification).

Common Errors

- Application of **INLETT** to a 2-D mesh system.
- Failure to specify the additional data values **NDATA**, **PTOT**, **TTOT**, **BETAR**, or **BETAT**.
- Radial-like direction of the mesh is not the j coordinate.
- **NDATA** less than 3, resulting in improper interpolation.
- **BETAR** and/or **BETAT** orientation incorrectly interpreted.
- **RAD**, **PTOT** and/or **TTOT** improperly normalized.

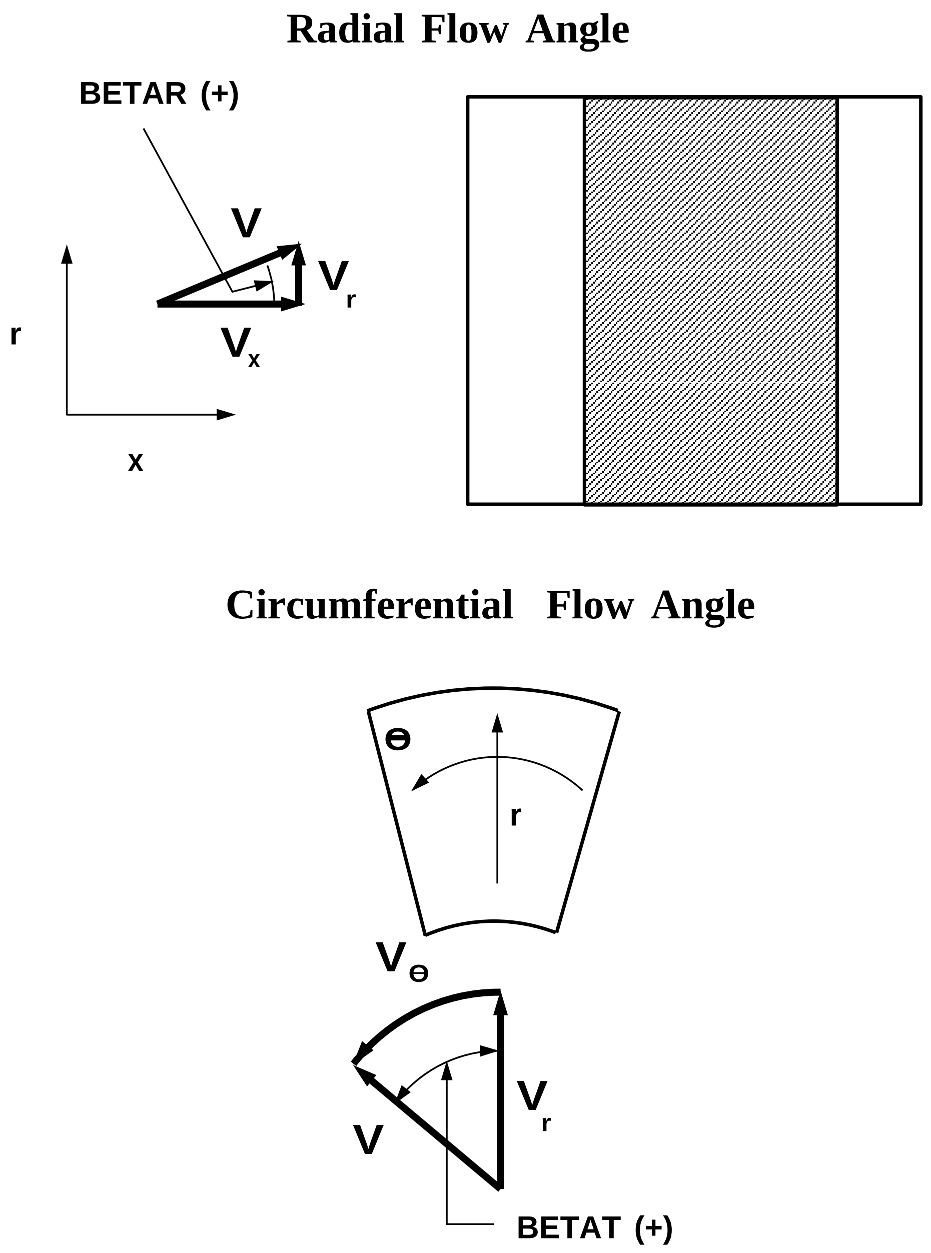
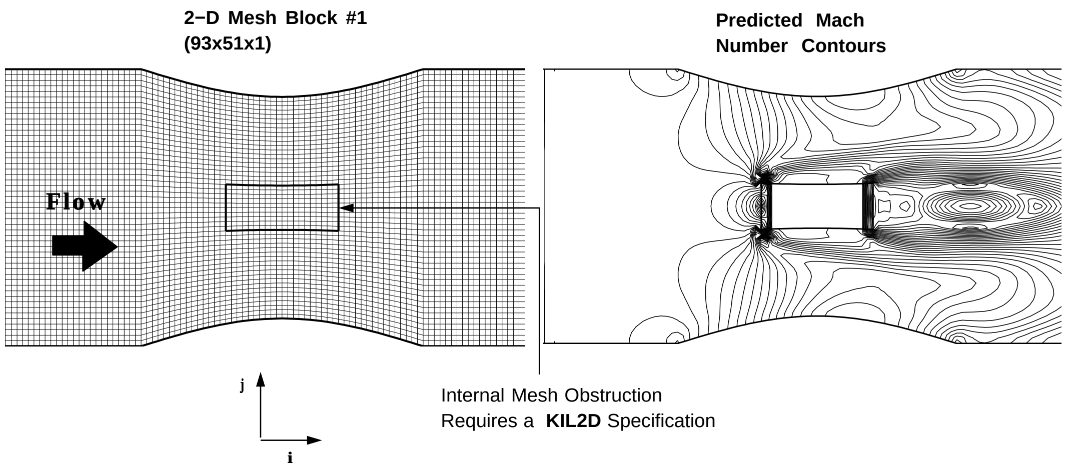


Figure 3.8: *ADPAC08* 2-D Single Block Mesh Structure Illustration

- Mesh/geometry not defined with the x axis as the centerline.

KIL2D

2-D Solution Kill Routine



Application

The **KIL2D** keyword is a tool to effectively neutralize or “kill” the time-marching solution over a segment of the computational domain for a two-dimensional mesh. The example graphic above illustrates a single block 2-D mesh system used to predict the flow through a converging/diverging nozzle system with a square-edged obstruction. Rather than construct a multiple block mesh system to treat this case (whereby the obstruction is essentially gridded as block boundaries), the **KIL2D** specification is used to neutralize the advancing solution within the obstruction, and boundary conditions are applied along the surface of the obstruction to predict this flow.

Boundary Data File Format

The boundary data file specification for the mesh interface indicated in the illustrative graphic for the **KIL2D** boundary condition is given below:

```
KIL2D   1  1  I  I  M  M  L  L  40  60  21  31   1   2  21  31   1   2
   LSTART   LEND
       40     60
```

Note that a complete **KIL2D** specification requires two additional lines following the **KIL2D** boundary data file specification line. Failure to properly specify the data in these additional lines is a common **KIL2D** specification error.

Description

In cases where a portion of a 2-D mesh does not represent a valid flow region, the **KIL2D** specification can be used, in conjunction with boundary conditions specified about the region to be "killed", to effectively remove a portion of a given mesh block from the computational domain. An example of this technique is illustrated in the illustrative graphic above. The figure depicts a single block mesh for the flow through a simple nozzle. Suppose that for whatever reason, the user wished to remove an internal rectangular portion of the mesh (as if there were an obstruction placed in the flowpath). This could be accomplished by subdividing the original mesh into several smaller pieces, and applying the appropriate boundary conditions along the outer boundaries of each block. This same configuration could also be modeled using the original mesh by invoking the **KIL2D** specification for the points inside the obstruction, and applying the proper boundary specifications along the obstruction internally on the single-block mesh. This boundary condition (although really, this is more than a boundary condition) requires the specification of additional data, as shown in the format descriptor above. The variable following the label **LSTART** indicates the starting index of the **LFACE1** coordinate direction (in the example above, this would be the **I** coordinate direction) for the region to be "killed". The variable following the label **LEND** indicates the final index in the **LFACE1** coordinate direction (again, the **I** coordinate in the example above) for the region to be “killed”. The remaining coordinate indices for the region to be “killed” are determined by the variables **M1LIM1, M1LIM2** for the **J** coordinate direction and **N1LIM1**, and **N1LIM2** for the **K** coordinate direction. The additional specification of the **LSTART, LEND** variables imply that the variables **L1LIM, L2LIM** are not used in this specification. The **KIL2D** routine fuctions by constantly resetting the flow variables inside the region to be killed to the initial values specified by the **RMACH** input variable. So, in effect, the solution is still being performed in the region to be killed, but the updated results are constantly reset to a uniform flow value. This routine is not without drawbacks. First of all, although the mesh points are effectively neutralized by the **KIL2D** specification, other routines such as the residual smoothing algorithm are unaltered, and under certain circumstances, this may cause poor convergence. It is also possible that divergence may occur within the “killed” cells in spite of the resetting procedure. The best advice is to manipulate block structures to eliminate the need for the use of the **KIL2D** routine, but the user should be aware that under dire circumstances this facility is available.

Restrictions/Limitations

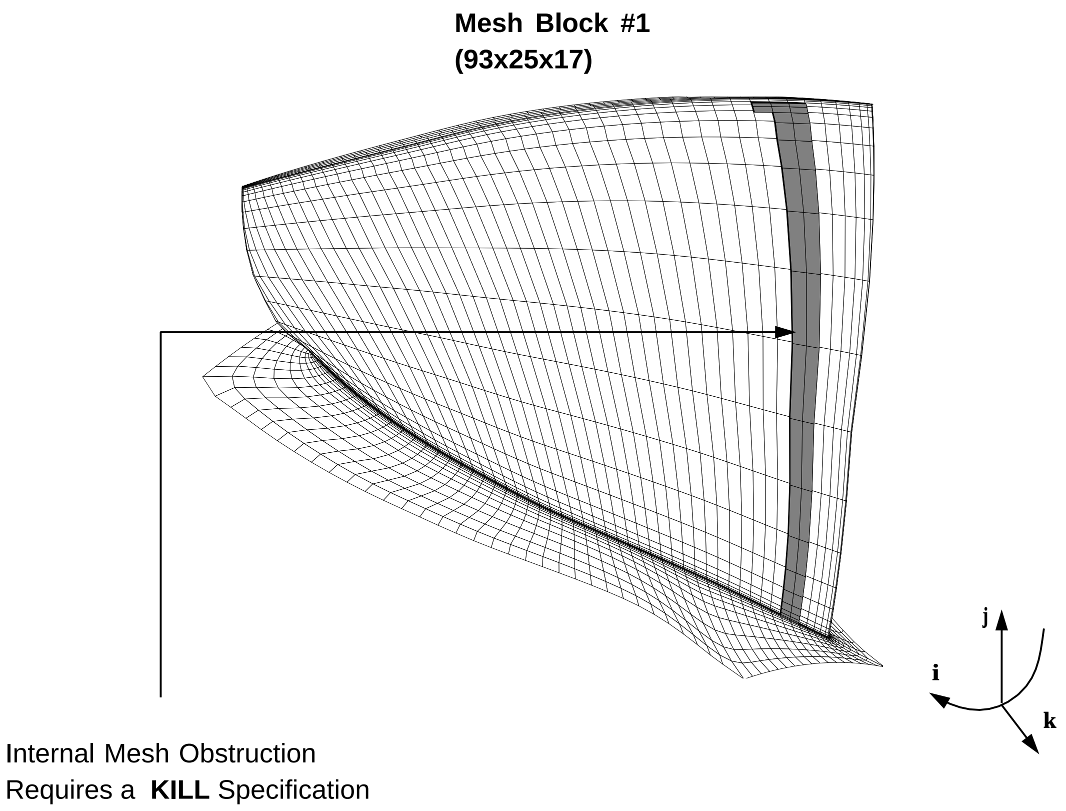
The **KIL2D** boundary specification is restricted to 2-D mesh surfaces (3-D mesh surfaces should use the **KILL** boundary specification).

Common Errors

- Application of **KIL2D** to a 3-D mesh system.
- Poor convergence due to residual smoothing across a “killed” region (The residual smoothing operator can be turned off through the **RESID** input variable, although the time step must be restricted (see variable **CFL**) to maintain numerical stability).
- Failure to specify the additional data values **LSTART**, **LEND**.

KILL

Solution Kill Routine



Application

The **KILL** keyword is a tool to effectively neutralize or "kill" the time-marching solution over a segment of the computational domain for a three-dimensional mesh. The example graphic above illustrates a single block 3-D O-type mesh system used to predict the flow through a turbomachinery compressor rotor blade passage with a surface-mounted square-edged obstruction. Rather than construct a multiple block mesh system to treat this case (whereby the obstruction is essentially gridded as block boundaries), the **KILL** specification is used to neutralize the advancing solution within the obstruction, and boundary conditions are applied along the surface of the obstruction to predict this flow.

Boundary Data File Format

The boundary data file specification for the mesh system indicated in the illustrative graphic for the **KILL** boundary condition is given below:

```
KILL   1  1  I  I  P  P  L  L  49  49   1  19   1   5   1  19   1   5
   LSTART   LEND
       49     51
KILL   1  1  I  I  P  P  L  L  49  49  19  21   1   5  19  21   1   5
   LSTART   LEND
       49     52
```

Note that a complete **KILL** specification requires two additional lines following the **KILL** boundary data file specification line. Failure to properly specify the data in these additional lines is a common **KILL** specification error.

Description

In cases where a portion of a 3-D mesh does not represent a valid flow region, the **KILL** specification can be used, in conjunction with boundary conditions specified about the region to be “killed”, to effectively remove a portion of a given mesh block from the computational domain. An example of this technique is illustrated in the illustrative graphic above. The figure depicts a single mesh block for the flow through a high speed rotor passage upon which surface instrumentation is mounted. The blockage associated with the surface instrumentation is incorporated into the solution through the application of the appropriate boundary conditions on the surface of the instrumentation, and by applying the **KILL** procedure to negate the flow variables of the cells within the instrumentation itself. It should be noted that this effect could be accomplished by subdividing the original mesh into several smaller pieces, and applying the appropriate boundary conditions along the outer boundaries of each block. This boundary condition (although really, this is more than a boundary condition) requires the specification of additional data, as shown in the format descriptor above. The variable following the label **LSTART** indicates the starting index in the **LFACE1** coordinate direction (in the example above, this would be the **I** coordinate direction) for the region to be "killed". The variable following the label **LEND** indicates the final index in the **LFACE1** coordinate direction (again, the **I** coordinate in the example above) for the region to be "killed". The remaining coordinate indices for the region to be "killed" are determined by the variables **M1LIM1, M1LIM2** for the **J** coordinate direction and **N1LIM1**, and **N1LIM2** for the **K** coordinate direction. The additional specification of the **LSTART, LEND** variables imply that the variables **L1LIM, L2LIM** are not used in this specification. The **KILL** routine fuctions by constantly resetting the flow variables inside the region to be killed to the initial values specified by the **RMACH** input variable. So, in effect, the solution is still being performed in the region to be killed, but the updated results are constantly reset to a uniform flow value. This routine is not without drawbacks. First of all, although the mesh points are effectively neutralized by the **KILL** specification, other routines such as the residual smoothing algorithm are unaltered, and under certain circumstances, this may cause poor convergence. It is also possible that divergence may occur within the "killed" cells in spite of the resetting procedure. The best advice is to manipulate block structures to eliminate the need for the use of thr **KILL** routine, but the user should be aware that under dire circumstances this facility is available.

Restrictions/Limitations

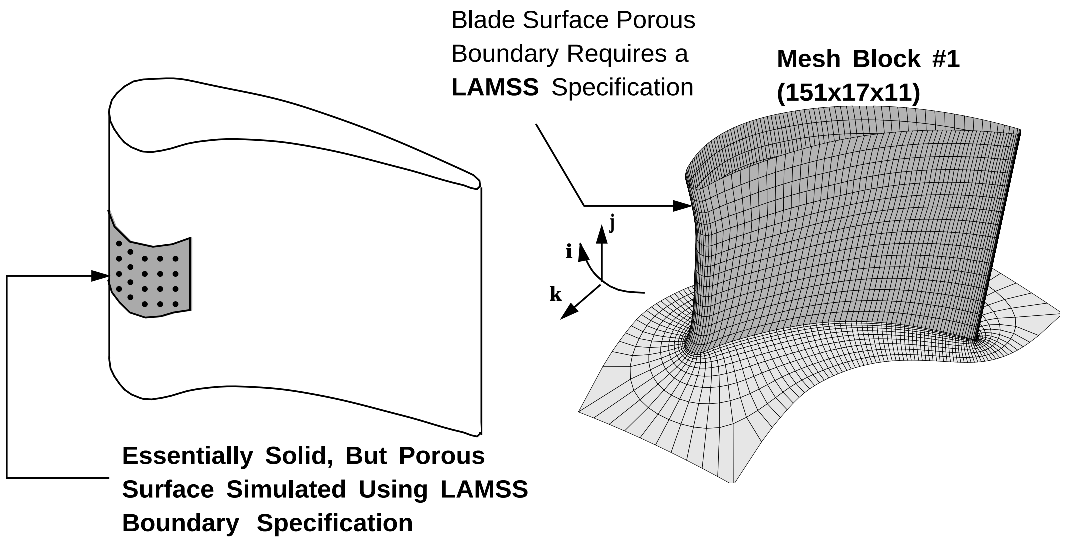
The **KILL** boundary specification is restricted to 3-D mesh surfaces (2-D mesh surfaces should use the **KIL2D** boundary specification).

Common Errors

- Application of **KILL** to a 2-D mesh system.
- Poor convergence due to residual smoothing across a “killed” region (The residual smoothing operator can be turned off through the **RESID** input variable, although the time step must be restricted (see variable **CFL**) to maintain numerical stability).
- Failure to specify the additional data values **LSTART**, **LEND**.

LAMSS

Porous Solid Surface Viscous No-Slip Boundary Condition


Blade Surface Porous Boundary Requires a **LAMSS** Specification
Mesh Block #1 (151x17x11)
j
i
k
Essentially Solid, But Porous Surface Simulated Using LAMSS Boundary Specification

Application

The **LAMSS** specification is used to impose a porous injection, no-slip boundary condition for solid surfaces used in a viscous flow solution. The graphic above illustrates a 3-D body-centered O-type mesh system for a turbine vane cascade. The **LAMSS** specification is used to simulate the effects of a fine array of discrete cooling holes (porous injection) which are too small to be individually gridded. The **LAMSS** provides a "smeared out" injection which essentially simulates the global effects of the individual cooling sites.

Boundary Data File Format

The boundary data file specifications for the hub and blade surfaces in the application described above and indicated in the illustrative graphic for the **LAMSS** boundary condition are given below:

```
LAMSS   1  1  K  K  P  P  I K  1   1  1 151  1  11  1 151  1  11
PT     TT    RPMLOC     TWALL   ARATIO
1.1    0.70  0.0        0.00    0.10
```

Note that a complete **LAMSS** specification requires two additional lines following the **LAMSS** boundary data file specification line. Failure to properly specify the data in these additional lines is a common **LAMSS** specification error.

Description

The **LAMSS** statement specifies that a solid surface viscous (no-slip) boundary condition is to be applied to the mesh surface specified by **LFACE1** on the block specified by **LBLOCK1**. The **LAMSS** boundary condition may be applied to either a rotating or non-rotating surface and may indicate a rotational speed which is different than the rotational speed of the mesh (**RPM**) to which the boundary condition is applied (the most common example of this type of application is a mesh embedded in a rotating blade passage with an endwall which is non-rotating). This boundary condition requires the specification of additional data, as shown in the boundary data format descriptor above. The first additional line following the **LAMSS** specification is assumed to be a label and may contain any information; however, for consistency it is recommended that the labels **PTOT**, **TTOT**, **RPMLOC**, **TWALL**, and **ARATIO** be used. The next line contains the values imposed for the variables **PTOT**, **TTOT**, **RPMLOC**, **TWALL**, and **ARATIO**. The value of the **PTOT** and **TTOT** variables represent the total pressure and total temperature of the injected flow. These variables are defined as:

$$(P_{total})_{non-dimensional} = \frac{P_{total}}{P_{ref}}$$

$$(T_{total})_{non-dimensional} = \frac{T_{total}}{P_{ref}}$$

The value of the **RPMWALL** variable is the desired solid wall dimensional rotational speed in revolutions per minute. This value is sign dependent and follows the orientation for rotation as described in Figure 3.9. The variable **TWALL** determines which type of temperature condition is applied to the surface. If **TWALL**=0.0, an adiabatic wall is assumed. For **TWALL**>0.0, a constant temperature surface with a nondimensional wall temperature of **TWALL** defined as:

$$(T_{wall})_{non-dimensional} = \frac{T_{wall}}{T_{ref}}$$

is imposed. (Here T_{ref} is the reference temperature imposed by the input file variable **TREF**. A value of **TWALL**<0.0 is not permitted. Naturally, poor convergence or solution divergence can occur if **RPMWALL** or **TWALL** suggest boundary values which are significantly different from the remainder of the flowfield. In such cases where this occurs, it is recommended that the solution be started with more conservative boundary values, and then restarted using the final boundary values. Finally, the value of the variable **ARATIO** represents the geometric "porosity" of the surface in the form of the ratio of open (injection) surface area to total surface area for the boundary segment being defined. In other words, if the porous surface has an injection area of .01 square inch per square inch of total surface, then **ARATIO** would be 0.01. **ARATIO** values less than zero or greater than 1.0 are not permitted.

Restrictions/Limitations

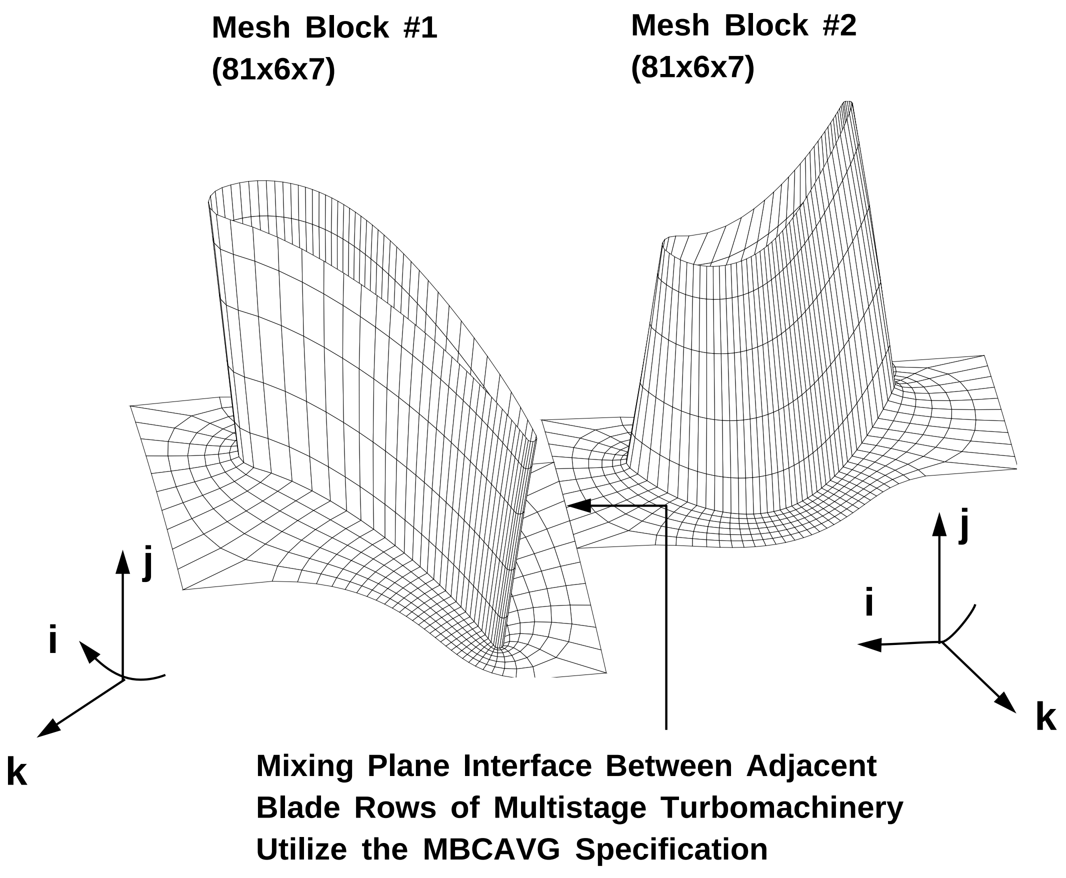
The boundary rotational speed imposed by the **LAMSS** boundary condition can only be nonzero when using the cylindrical coordinate solution algorithm in the *ADPAC08* code. When using the Cartesian coordinate solution algorithm **FCART,FCARB**= 1.0, the boundary rotational speed must be zero (**RPMWALL**= 0.0 when **FCART,FCARB**= 1.0). Refer to the Chapter on input file parameters for a description of **TREF**, **RPM**, **FCARB**, and **FCART**.

Common Errors

- Incorrect sign for value of boundary rotational speed **RPMWALL**.
- Attempt to utilize a nonzero boundary rotational speed with the Cartesian coordinate solution algorithm.
- **ARATIO** value less than 0.0 or greater than 1.0.
- **PTOT** value significantly different than freestream.
- **TTOT** value significantly different than freestream.
- **TWALL** value significantly different than freestream.

MBCAVG

Multiple Block Circumferential Averaging Routine for Multiple Blade Row Turbomachines


Mesh Block #1
(81x6x7)
Mesh Block #2
(81x6x7)
j
i
k
j
i
k
Mixing Plane Interface Between Adjacent
Blade Rows of Multistage Turbomachinery
Utilize the MBCAVG Specification

Application

The **MBCAVG** specification is used in applications involving neighboring relatively rotating blade rows which may consist of one or more mesh blocks. The **MBCAVG** specification permits time-averaged interconnection between these adjacent, blade row local mesh systems based on the "mixing plane" approximation discussed in Chapter 2.0. The sample graphic illustrates the application of the **MBCAVG** boundary condition for the case of a single stage turbine, whereby a single mesh block is used to represent a single blade passage for each blade row in the turbine stage, and the **MBCAVG** boundary routine is used to perform the mixing plane (circumferential/time-averaged) coupling of the relatively rotating blade rows.

Boundary Data File Format:

The boundary data file specifications for the mesh interfaces indicated in the illustrative graphic for the **MBCAVG** boundary condition are given below. Note that block 1 requires multiple specifications due to the location of the O-grid cut line.

```
MBCAVG    1  2  K  K  M  M  I  J   7   7   1   6   1   6  36  46   1   6
NSEGS
     1
LBLOCK2B  LFACE2B  LDIR2B   L2LIMB  M2LIM1B  M2LIM2B  N2LIM1B  N2LIM2B
2         K        M        7       36       46       1        6
MBCAVG    1  2  K  K  M  M  I  J   7   7  76  81   1   6  36  46   1   6
NSEGS
     1
LBLOCK2B  LFACE2B  LDIR2B   L2LIMB  M2LIM1B  M2LIM2B  N2LIM1B  N2LIM2B
2         K        M        7       36       46       1        6
MBCAVG    2  1  K  K  M  M  I  J   7   7  36  46   1   6   1   6   1   6
NSEGS
     2
LBLOCK2B  LFACE2B  LDIR2B   L2LIMB  M2LIM1B  M2LIM2B  N2LIM1B  N2LIM2B
1         K        M        7        1        6       1        6
1         K        M        7       76       81       1        6
```

Note that a complete **MBCAVG** specification generally requires at least two **MBCAVG** statement lines in the boundary data file for each mesh interface. In the example above, the first specification provides the interblock communication for block 1 from block 2, and the second specification provides the communication for block 2 from block 1. It is a common error to underspecify an **MBCAVG** boundary by only providing a single line per interface.

Description:

The **MBCAVG** specification provides a "circumferential mixing plane" mesh block communication scheme for steady state (time-averaged) analysis of multiple blade row turbomachines. The **MBCAVG** operator permits the specification of multiple neighboring blocks upon which the circumferential averaging is applied to provide boundary data for the current block of interest. This multiple block averaging scheme permits the use of **MBCAVG** for body-centered mesh systems (see the illustrative graphic above) and also for multiple blade passage representations of neighboring blade rows. Due to the complex nature of the circumferential averaging operator, this boundary condition is restricted to specific mesh configurations. The following chart describes the permitted mesh configurations for the **MBCAVG** specification:

MBCAVG Boundary Specification Mesh Coordinate Restrictions

LFACE1 (Block #1 Face)	LFACE2 (Block #2 Face)	Circumferential Coordinate Direction	Grids Must be Aligned in this Coordinate
I	I or K	K or I	J
J	J only	K	I
K	I or K	K or I	J

A second mesh restriction is that the interface separating two adjacent blade rows must be a surface of revolution, and that meshes along this interface have common axial and radial grid distributions. This restriction simplifies the averaging scheme provided by the **MBCAVG** specification.

The **MBCAVG** boundary condition requires the specification of additional data, as shown in the format descriptor above. The variable following the label **NBLKS** defines the number of neighboring mesh block surfaces from which the circumferentially averaged data is obtained. In the illustrative graphic above, this value is simply 1 for the upstream inter-blade row boundaries, but is 2 for the downstream inter-blade row boundary because of the fact that the matching boundary of the upstream blade row is composed of two distinct mesh segments even though it is taken from a single mesh block. The next line following the **NSEGS** variable is a label indicating the variables which must be input for each of the **NSEGS** segments in the mixing plane. The variables **LBLOCK2B**, **LFACE2B**, **LDIR2B**, **L2LIMB**, **M2LIM1B**, **M2LIM2B**, **N2LIM1B**, and **N2LIM2B** represent the values of **LBLOCK2**, **LFACE2**, **LDIR2**, **L2LIM**, **M2LIM1**, **M2LIM2**, **N2LIM1**, and **N2LIM2** (see the beginning of this section for an explanation of these variables) for *each* of the individual **NSEGS** segments used in the mixing plane construction. The segments may be specified in any order.

Restrictions/Limitations

The **BCPRR** boundary specification is restricted to mesh interfaces which lie on a common surface (no significant overlap), and have common axial and radial mesh coordinates. The mesh must obey the coordinate restrictions outlined in the description above. The **MBCAVG** procedure is only applicable to 3-D mesh systems.

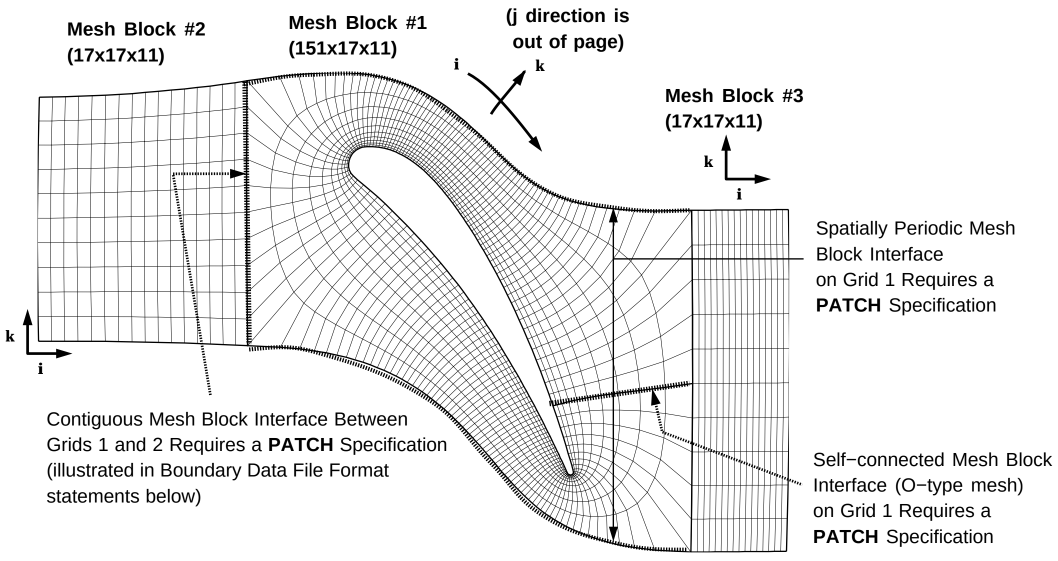
Common Errors

- Failure to provide 2 or more **MBCAVG** statements for each inter-blade row interface.
- Incorrectly specified or misaligned extents of boundary regions (values of **M1LIM1, M1LIM2, N1LIM1, N1LIM2, M2LIM1B, M2LIM2B, N2LIM1B, N2LIM2B** do not correctly define the interface extents on blocks **LBLOCK1** and **LBLOCK2B**)

- Attempt to use **MBCAVG** on a 2-D mesh block.
- Meshes do not obey the mesh coordinate restrictions listed in the description above.
- Meshes have dissimilar axial and radial coordinates at the interface.
- Application of **MBCAVG** to mesh interfaces which do not share a common surface, or which have excess overlap.
- Application of **MBCAVG** to Cartesian solution mesh systems.

PATCH

Contiguous Mesh Block Interface Patching Scheme



Application

The **PATCH** specification is used in any application involving neighboring mesh blocks with a contiguous (common mesh points) interface. The graphic above illustrates a **PATCH** connection between mesh blocks in a combination O-H mesh system for a turbine vane cascade. The **PATCH** boundary specification is used to provide block-to-block communication between mesh blocks #1 and #2, and mesh blocks #1 and #3, as well as providing periodic flow boundary conditions for blocks #1, #2, and #3. In addition, the **PATCH** routine is used to provide aerodynamic communication across the O-mesh slit for mesh block #1. The **PATCH** boundary condition is perhaps the most common specification resulting from the use of the multiple blocked mesh capabilities of the *ADPAC08* code.

Boundary Data File Format

The boundary data file specifications for the mesh interface indicated in the illustrative graphic for the **PATCH** boundary condition are given below:

```
PATCH 1 1 K K M M I J  11  11   6  71  1 17 146  81  1 17  Blk #1 Per
PATCH 1 1 K K M M I J  11  11  81 146  1 17  71   6  1 17  Blk #1 Per
PATCH 2 2 K K P M I J   1  11   1  17  1 17   1  17  1 17  Blk #2 Per
PATCH 2 2 K K M P I J  11   1   1  17  1 17   1  17  1 17  Blk #2 Per
PATCH 3 3 K K P M I J   1  11   1  17  1 17   1  17  1 17  Blk #3 Per
PATCH 3 3 K K M P I J  11   1   1  17  1 17   1  17  1 17  Blk #3 Per
PATCH 1 1 I I M P J K 151   1   1  17  1 11   1  17  1 11  Blk #1 O-Grid
PATCH 1 1 I I M P J K   1 151   1  17  1 11   1  17  1 11  Blk #1 O-Grid
PATCH 1 2 K I M M J I  11  17  71  81  1 17   1  17  1 11  Blks #1-#2
PATCH 2 1 I K M M K J  17  11   1  17  1 11  71  81  1 17  Blks #1-#2
PATCH 1 3 K I M P J I  11   1   1   6  1 17   1  17  6  1  Blks #1-#3
PATCH 3 1 I K P M K J   1  11   1  17  1  6   6   1  1 17  Blks #1-#3
PATCH 1 3 K I M P J I  11   1 146 151  1 17   1  17 11  6  Blks #1-#3
PATCH 3 1 I K P M K J   1  11   1  17  6 11 151 146  1 17  Blks #1-#3
```

Note that a complete **PATCH** specification generally requires two **PATCH** statement lines in the boundary data file. For any two grid blocks (1 and 2 for example), the first specification provides the interblock communication for block 1 from block 2, and the second specification provides the communication for block 2 from block 1. It is a common error to underspecify a **PATCH** boundary by only providing a single line per interface.

Description

The **PATCH** statement is utilized to provide direct block to block communication between mesh blocks with contiguous grid points. This is perhaps the most common, and most useful of the boundary condition specifications, and therefore, a lengthy discussion is given to complete this description. For many complicated geometries requiring a multiple block mesh system, a common approach is to generate mesh systems with coincident mesh points along all, or at least part of the mesh block interfaces. This property is henceforth referred to as a contiguous mesh block interface (coincident mesh points). By default, the boundary condition specification **must** have a one to one correspondence between mesh points in block **LBLOCK1** and mesh points in block **LBLOCK2**. This type of boundary is particularly effective in the finite-volume flow solver due to the fact that local and global conservation of the flow variables can be accomplished without special treatment, by direct substitution of the neighboring block flow variables into the phantom cells of the block of interest. The **PATCH** boundary condition performs this direct substitution between blocks to provide an aerodynamic communication between neighboring blocks with a contiguous interface. A **PATCH** specification can also be imposed connecting a block to itself. In fact, this is the manner by which spatial periodicity is enforced in many cases, including the Standard Configurations described in Chapter 5. The **PATCH** boundary condition requires no additional data beyond the initial specification line, but does require the proper specification of the variables **LSPEC1** and **LSPEC2**. For boundary conditions involving more than one mesh block (such as **PATCH**), it is possible that the connection between blocks may involve communication between different grid surfaces (for example, an i=constant mesh face in **LBLOCK1** connects to a j=constant mesh face in **LBLOCK2**) and that the remaining indices in block **LBLOCK2** correspond to different coordinates in block **LBLOCK1**. The specification of the variables **LSPEC1, LSPEC2** serve to eliminate any confusion between contiguous boundary patches involving dissimilar mesh coordinates. In every case, when a particular coordinate direction is specified by the variable **LFACE1**, the remaining coordinate indices defining the extent of the patch on **LFACE1** are specified in their "natural" (i, j, k) order. For example, if **LFACE1** is an i=constant mesh surface, then the variables **M1LIM1, M1LIM2** control the indices in the j coordinate direction and the variables **N1LIM1, N1LIM2** control the indices in the k coordinate direction. Similarly, if **LFACE2** is a k=constant mesh surface, then the variables **M2LIM1, M2LIM2** control the indices in the i coordinate direction and the variables **N2LIM1, N2LIM2** control the indices in the j coordinate direction, and so on. Now, in order to relate the coordinate indices in block **LBLOCK2** with the indices specified in block **LBLOCK1**, the special terms **LSPEC1** and **LSPEC2** are utilized. The variables **LSPEC1** and **LSPEC2** should be defined as either **I, J**, or **K**, based on the connection scheme between the two blocks. The **LSPEC1** variable should define the coordinate direction in block **LBLOCK1** which corresponds to the first remaining coordinate in block **LBLOCK2** (whose range is defined by **M2LIM1, M2LIM2**), and the **LSPEC2** variable should define the coordinate direction in block **LBLOCK1** which corresponds to the second remaining coordinate in block **LBLOCK2** (whose range is defined by **N2LIM1, N2LIM2**). The **PATCH** specification may also be used for two-dimensional meshes as long as the third coordinate direction (k) limits **N1LIM1**, **N1LIM2**, and **N2LIM1**, **N2LIM2** are "1" and "2", respectively (2-D patches are specified as if the mesh were actually 2 cells deep in the k direction).

Restrictions/Limitations

The **PATCH** boundary specification is restricted to mesh interfaces which have a one to one mesh point correspondance. To maintain the conservative property of the governing equations, the **PATCH** routine assumes that the mesh points between the 2 blocks of interest are either contiguous, or share a spatially periodic relationship, and it is left to the user to verify that this is so.

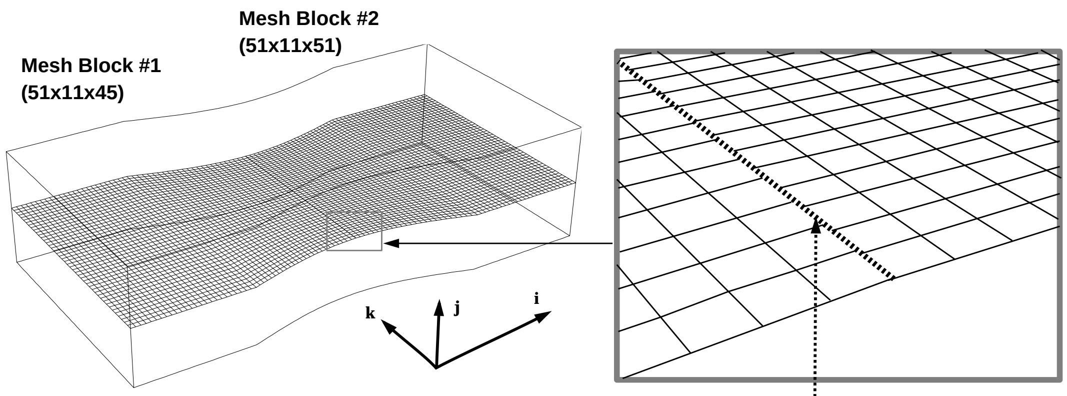
Common Errors

- Failure to provide 2 **PATCH** statements for each interface.
- Incorrectly specified or misaligned extents of boundary regions (values of **M1LIM1, M1LIM2, N1LIM1, N1LIM2, M2LIM1, M2LIM2, N2LIM1, N2LIM2** do not correctly define the interface extents on blocks **LBLOCK1** and **LBLOCK2**).
- Incorrectly specified block coordinate direction relationships (values of **LSPEC1, LSPEC2** do not correctly define the coordinate connection scheme between block **LBLOCK1** and block **LBLOCK2**).
- **PATCH** boundary specification for a periodic boundary is applied to a nonperiodic mesh.

- **PATCH** boundary specification applied to a spatially periodic Cartesian geometry using the cylindrical coordinate solution scheme or vice versa (results in incorrect spatial periodicity relationships) The **PATCH** boundary specifications for Cartesian geometries must use the Cartesian solution algorithm in *ADPAC08* (see input variable **FCART**).

PINT

Non-Contiguous Mesh Block Interface Patching Scheme



Non−Contiguous Mesh Block Interface Between Grids 1 and 2 Requires a **PINT** Specification (illustrated in Boundary Data File Format statements below)

Application

The **PINT** specification is used in any application involving neighboring mesh blocks which share a common mating surface (either contiguous or non-contiguous). The example graphic above illustrates a two-dimensional plane of a two block 3-D mesh system used to predict the flow through a converging/diverging nozzle. The mesh points at the interface between the two grids (near the nozzle throat) are non-contiguous, and therefore, the **PINT** specification is used to provide communication between the adjacent mesh blocks.

Boundary Data File Format

The boundary data file specifications for the mesh interface indicated in the illustrative graphic for the **PINT** boundary condition are given below:

```
PINT    1  2  I  I  M  P  L  L  51   1   1  11   1  45   1  11   1  51
PINT    2  1  I  I  P  M  L  L   1  51   1  11   1  51   1  11   1  45
```

Note that a complete **PINT** specification generally requires two **PINT** statement lines in the boundary data file. In the example above, the first specification provides the interblock communication for block 1 from block 2, and the second specification provides the communication for block 2 from block 1. It is a common error to underspecify a **PINT** boundary by only providing a single line per interface.

Description

The **PINT** boundary statement provides a means for block to block communication for cases involving neighboring meshes which share a common surface, but not necessarily common grid points along a block boundary (meshes with contiguous mesh points should use the **PATCH** boundary specification). The **PINT** specification instructs the *ADPAC08* code to perform a weighted interpolation to determine the appropriate flow variables for the phantom cells, based on the non-contiguous data structure of the neighboring mesh. An example of this type of boundary is given in the illustrative graphic. The bounding surfaces of each block should lie on a common surface (no significant overlap). The interpolation scheme used in the **PINT** specification is not conservative, and therefore the solution accuracy can be degraded by this procedure. During code execution, the first time the **PINT** specification is encountered, the code initiates a search to determine the interpolation stencil for the given array of points in block **LBLOCK1** based on the data in block **LBLOCK2**. This stencil is then saved to eliminate the search routine at every application of **PINT**. In order to provide storage for the interpolation stencil information, a separate array system based on the dimensioning parameter **NRAINT** (see Section 3.3) is utilized. The **PINT** boundary condition requires no additional data beyond the initial specification line, but does require some extra care when used. The primary precaution is that the **PINT** procedure is based entirely on a simplified interpolations scheme, and hence, does not maintain either global or local conservation of flow variables across the mesh interface.

Restrictions/Limitations

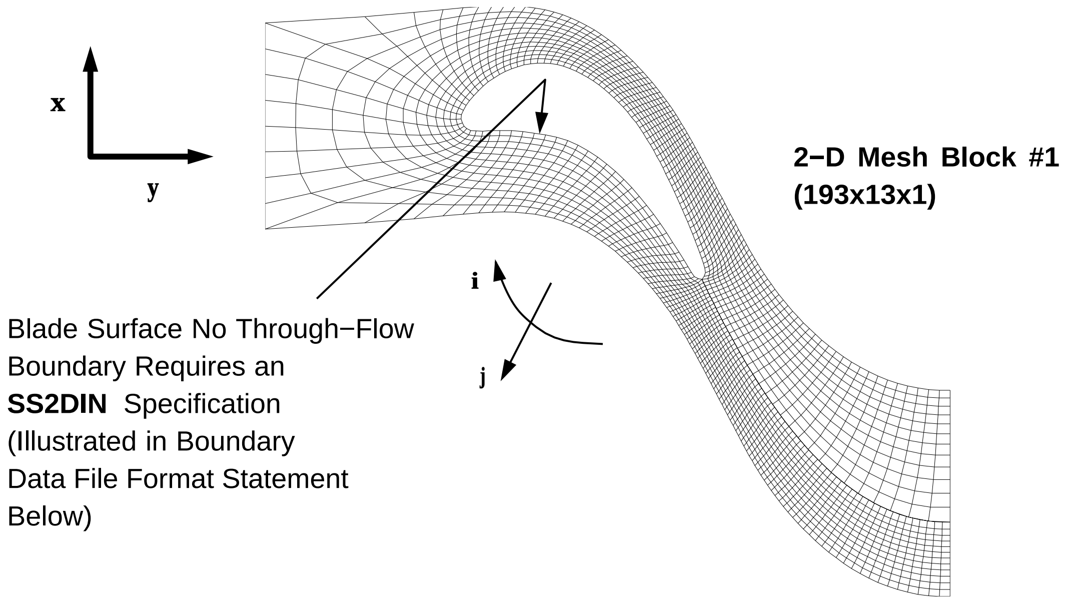
The **PINT** boundary specification is restricted to mesh interfaces which lie on a common surface (no significant overlap). The **PINT** procedure is only applicable to 3-D mesh systems. **PINT** can not be used across multiple processors in a parallel computing environment.

Common Errors

- Failure to provide 2 **PINT** statements for each interface.
- Incorrectly specified or misaligned extents of boundary regions (values of **M1LIM1, M1LIM2, N1LIM1, N1LIM2, M2LIM1, M2LIM2, N2LIM1, N2LIM2** do not correctly define the interface extents on blocks **LBLOCK1** and **LBLOCK2**).
- Attempt to use **PINT** for a periodic boundary (no special spatial periodicity arrangement is available in **PINT**.
- Attempt to use **PINT** on a 2-D mesh block.
- Failure to provide enough storage for the **PINT** interpolation stencils via the **NRAINT** parameter.
- Application of **PINT** to mesh interfaces which do not share a common surface, or which have excess overlap.
- Attempt to use **PINT** across multiple processors in a parallel computing environment.

SS2DIN

2-D Solid Surface Inviscid No Through-Flow Boundary Condition



Application

The **SS2DIN** specification is used to impose a no through-flow inviscid solid surface condition for any solid surface in a 2-D solution. The illustrative graphic above depicts a 2-D C-type Cartesian mesh system for a planar turbine airfoil cascade. The **SS2DIN** descriptor is applied to denote the airfoil no through-flow surface boundary condition. Applications for 2-D turbmomachinery calculations are typically the endwall (axisymmetric flow) or airfoil (Cartesian flow) surfaces.

Boundary Data File Format

The boundary data file specification for the mesh boundary indicated in the illustrative graphic for the **SS2DIN** boundary condition is given below:

```
SS2DIN   1  1  J  J  P  P  I K  1  1  65 161  1   2 65 161  1   2
```

No additional data beyond the boundary data file descriptor is required.

Description

The **SS2DIN** statement specifies that a solid surface inviscid (no through-flow) boundary condition is to be applied to the mesh surface specified by **LFACE1** on the block specified by **LBLOCK1**. The **SS2DIN** procedure is the 2-D equivalent of **SSIN**. The **SS2DIN** boundary condition may be applied to either rotating or non-rotating surfaces. The rotational speed of the boundary is irrelevant for an inviscid surface on a properly defined mesh (either the boundary rotates with the mesh, or, in the case where the rotational speeds of the mesh and boundary differ, there is no difference in the aplication of an inviscid surface boundary condition). **SS2DIN** may be applied for either cylindrical or Cartesian solution meshes (see the description of the input variable **FCART**). The **SS2DIN** algorithm imposes no flow normal to the local mesh surface, but permits tangential velocity components at the boundary. The current **SS2DIN** algorithm is based on a loose specification of the local static pressure ($\frac{\partial p}{\partial n} = 0$) and is known to introduce some nonphysical loss. However, it has been the authors experience that this formulation provides the best mix of accuracy and reliability for most applications. It should be noted that the **SS2DIN** boundary condition is also very useful as a method of imposing a symmetry plane in a solution for geometries which possess spatial symmetry. Naturally, the mesh must be generated in a manner which represents this spatial symmetry as well.

Restrictions/Limitations

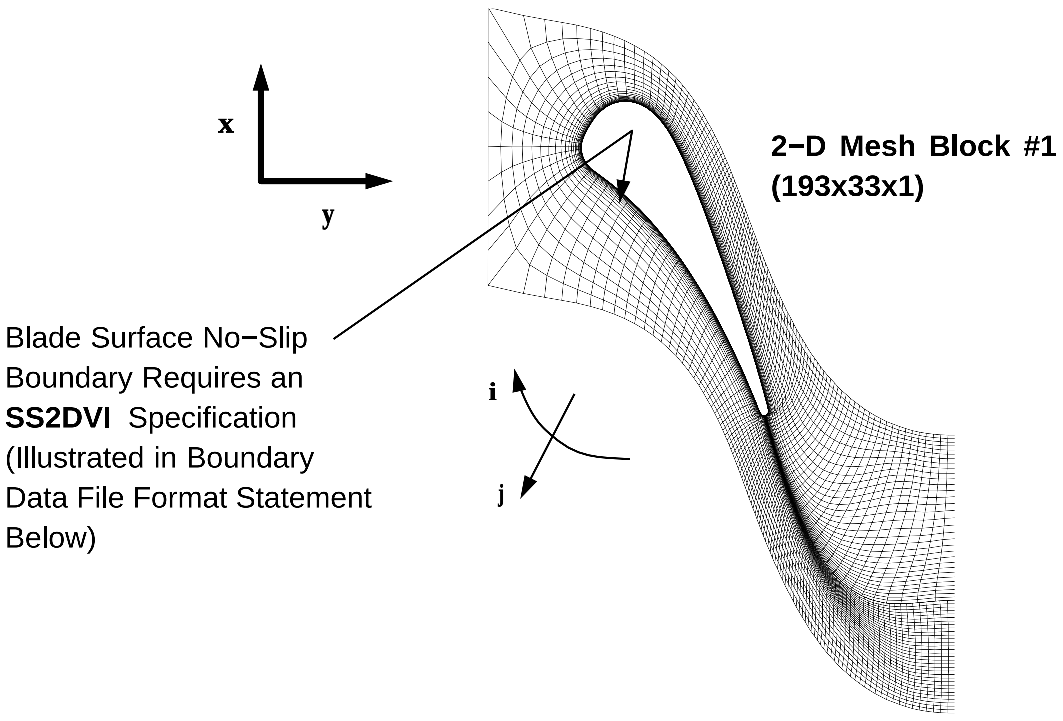
The **SS2DIN** boundary specification is restricted to 2-D mesh surfaces (3-D mesh surfaces should use the **SSIN** boundary specification).

Common Errors

- Application of **SS2DIN** to a 3-D mesh system.
- Application of **SS2DIN** as a symmetry plane condition for a mesh which does not represent a spatially symmetric geometry.

SS2DVI

2-D Solid Surface Viscous No-Slip Boundary Condition



Application

The **SS2DVI** specification is used to impose a no-slip boundary condition for any solid surface used in a 2-D viscous flow solution. The example graphic above illustrates a 2-D C-type mesh system used to predict the flow through a planar 2-D turbine vane cascade. Applications for 2-D turbmomachinery calculations are typically the endwall surfaces (both rotating and non-rotating surfaces) for an axisymmetric 2-D solution or non-rotating solid surfaces in

Cartesian 2-D solutions.

Boundary Data File Format:

The boundary data file specifications for the mesh interface indicated in the illustrative graphic for the **SS2DVI** boundary condition are given below:

```
SS2DVI    1  1  J  J  P  P  I K  1   1 65 161  1   2 65 161  1   2
RPMWALL  TWALL
0.0      0.0
```

Note that a complete **SS2DVI** specification requires two additional lines following the **SS2DVI** boundary data file specification line. Failure to properly specify the data in these additional lines is a common **SS2DVI** specification error.

Description

The **SS2DVI** statement specifies that a solid surface viscous (no-slip) boundary condition is to be applied to the mesh surface specified by **LFACE1** on the block specified by **LBLOCK1**. The **SS2DVI** procedure is the 2-D equivalent of **SSVI**. The **SS2DVI** boundary condition may be applied to either a rotating or non-rotating surfaces and may indicate a rotational speed which is different than the rotational speed of the mesh (**RPM**) to which the boundary condition is applied. This boundary condition requires the specification of additional data, as shown in the boundary data format descriptor above. The first additional line following the **SS2DVI** specification is assumed to be a label and may contain any information; however, for consistency it is recommended that the labels **RPMWALL** and **TWALL** be used. The next line contains the values imposed for the variables **RPMWALL** and **TWALL**. The value of the **RPMWALL** variable is the desired solid wall dimensional rotational speed in revolutions per minute. This value is sign dependent and follows the orientation for rotation as described in Figure 3.9. The variable **TWALL** determines which type of temperature condition is applied to the surface. If **TWALL**=0.0, an adiabatic wall is assumed. For

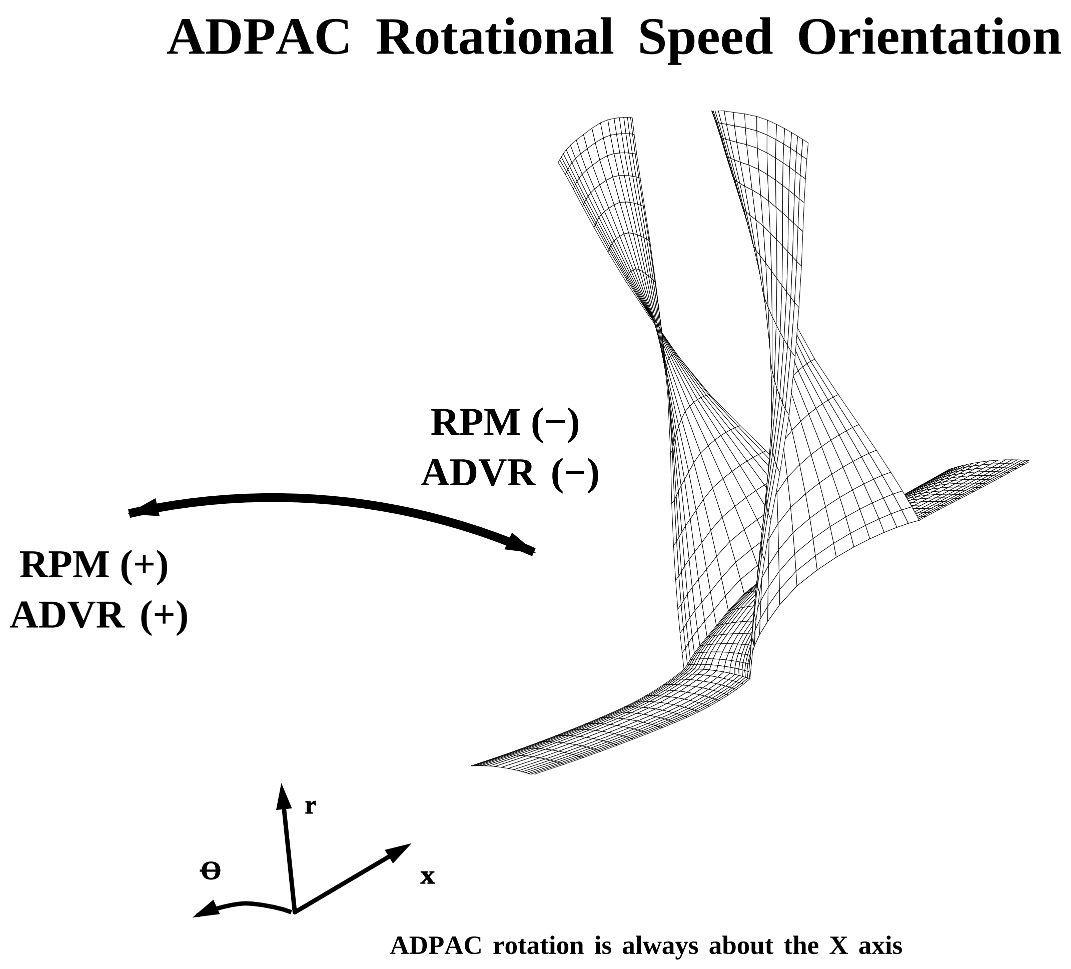

ADPAC Rotational Speed Orientation
RPM (−)
ADVR (−)
RPM (+)
ADVR (+)
r
θ
x
ADPAC rotation is always about the X axis

Figure 3.9: *ADPAC08* Rotational Speed Orientation Illustration

TWALL>0.0, a constant temperature surface with a nondimensional wall temperature of **TWALL** defined as:

$$(T_{wall})_{non-dimensional} = \frac{T_{wall}}{T_{ref}}$$

is imposed. (Here T_{ref} is the reference temperature imposed by the input file variable **TREF**. A value of **TWALL**<0.0 is not permitted. Naturally, poor convergence or solution divergence can occur if **RPMWALL** or **TWALL** suggest boundary values which are significantly different from the remainder of the flowfield. In such cases where this occurs, it is recommended that the solution be started with more conservative boundary values, and then restarted using the final boundary values.

Restrictions/Limitations

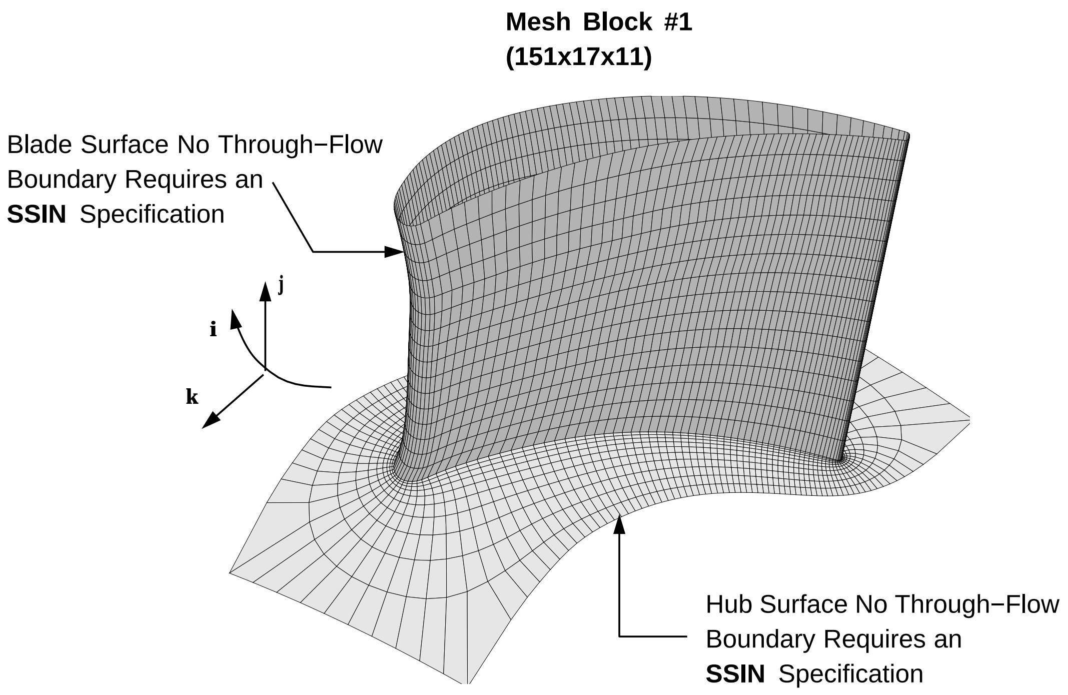
The **SS2DVI** boundary specification is restricted to 2-D mesh surfaces (3-D mesh surfaces should use the **SSVI** boundary specification). The boundary rotational speed imposed by the **SS2DVI** boundary condition can only be non-zero when using the cylindrical coordinate solution algorithm in the *ADPAC08* code, and may only be applied to axisymmetric 2-D meshes. When using the Cartesian coordinate solution algorithm (**FCART**= 1.0), the boundary rotational speed must be zero (**RPMWALL**= 0.0 when **FCART**= 1.0). Refer to the chapter on input file parameters for a description of **TREF**, **RPM**, and **FCART**.

Common Errors

- Incorrect sign for value of boundary rotational speed **RPMWALL**.
- Attempt to utilize a non-zero boundary rotational speed with the Cartesian coordinate solution algorithm.
- Application of **SS2DVI** to a 3-D mesh system.

SSIN

Solid Surface Inviscid No-Through-Flow Boundary Condition


Mesh Block #1
(151x17x11)
Blade Surface No Through−Flow Boundary Requires an **SSIN** Specification
j
i
k
Hub Surface No Through−Flow Boundary Requires an **SSIN** Specification

Application

The **SSIN** specification is used to impose a no-through-flow inviscid solid surface condition for any solid surface in a solution. The graphic above illustrates a 3-D body-centered O-type mesh system for a turbine vane cascade. For turbmomachinery calculations, the **SSVI** specification is normally used to define the blade and endwall surfaces (both rotating and non-rotating surfaces may be described).

Boundary Data File Format

The boundary data file specification for the mesh boundary indicated in the illustrative graphic for the **SSIN** boundary condition is given below:

```
SSIN   1  1  J  J  P  P  I K  1   1  1 151  1  11  1 151  1  11
SSIN   1  1  K  K  P  P  I K  1   1  1 151  1  17  1 151  1  17
```

No additional data beyond the boundary data file descriptor is required.

Description

The **SSIN** statement specifies that a solid surface inviscid (no through-flow) boundary condition is to be applied to the mesh surface specified by **LFACE1** on the block specified by **LBLOCK1**. The **SSIN** boundary condition may be applied to either rotating or non-rotating surfaces. The rotational speed of the boundary is irrelevant for an inviscid surface on a properly defined mesh (either the boundary rotates with the mesh, or, in the case where the rotational speeds of the mesh and boundary differ, there is no difference in the application of an inviscid surface boundary condition). The **SSIN** algorithm imposes no flow normal to the local mesh surface, but permits tangential velocity components at the boundary. The current **SSIN** algorithm is based on a loose specification of the local static pressure ($\frac{\partial p}{\partial n} = 0$) and is known to introduce some nonphysical loss. However, it has been the authors experience that this formulation provides the best mix of accuracy and reliability for most applications. It should be noted that the **SSIN** boundary condition is also very useful as a method of imposing a symmetry plane in a solution for geometries which possess spatial symmetry. Naturally, the mesh must be generated in a manner which represents this spatial symmetry as well.

Restrictions/Limitations

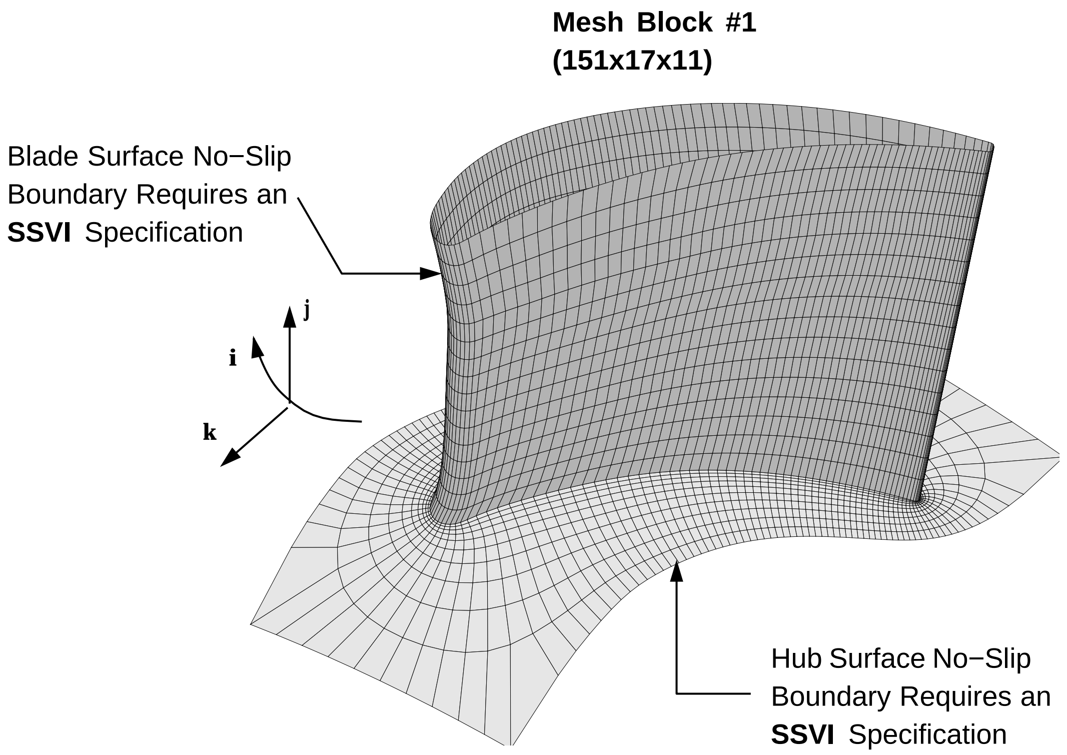
The **SSIN** boundary specification is restricted to 3-D mesh surfaces (2-D mesh surfaces should use the **SS2DIN** boundary specification).

Common Errors

- Application of **SSIN** to a 2-D mesh system.
- Application of **SSIN** as a symmetry plane condition for a mesh which does not represent a spatially symmetric geometry

SSVI

Solid Surface Viscous No-Slip Boundary Condition


Mesh Block #1
(151x17x11)
Blade Surface No−Slip Boundary Requires an **SSVI** Specification
j
i
k
Hub Surface No−Slip Boundary Requires an **SSVI** Specification

Application

The **SSVI** specification is used to impose a no-slip boundary condition for solid surfaces used in a viscous flow solution. The graphic above illustrates a 3-D body-centered O-type mesh system for a turbine vane cascade. For turbmomachinery calculations, the **SSVI** specification is normally used to define the blade and endwall surfaces (both rotating and non-rotating surfaces may be described).

Boundary Data File Format

The boundary data file specifications for the hub and blade surfaces in the application described above and indicated in the illustrative graphic for the **SSVI** boundary condition are given below:

```
SSVI    1   1   J   J   P   P   I K   1    1   1 151   1   11   1 151   1   11
RPMWALL   TWALL
0.0       0.0
SSVI    1   1   K   K   P   P   I K   1    1   1 151   1   17   1 151   1   17
RPMWALL   TWALL
0.0       0.0
```

Note that a complete **SSVI** specification requires two additional lines following the **SSVI** boundary data file specification line. Failure to properly specify the data in these additional lines is a common **SSVI** specification error.

Description

The **SSVI** statement specifies that a solid surface viscous (no-slip) boundary condition is to be applied to the mesh surface specified by **LFACE1** on the block specified by **LBLOCK1**. The **SSVI** boundary condition may be applied to either a rotating or non-rotating surface and may indicate a rotational speed which is different than the rotational speed of the mesh (**RPM**) to which the boundary condition is applied (the most common example of this type of application is a mesh embedded in a rotating blade passage with an endwall which is non-rotating). This boundary condition requires the specification of additional data, as shown in the boundary data format descriptor above. The first additional line following the **SSVI** specification is assumed to be a label and may contain any information; however, for consistency it is recommended that the labels **RPMWALL** and **TWALL** be used. The next line contains the values imposed for the variables **RPMWALL** and

TWALL. The value of the **RPMWALL** variable is the desired solid wall dimensional rotational speed in revolutions per minute. This value is sign dependent and follows the orientation for rotation as described in Figure 3.9. The variable **TWALL** determines which type of temperature condition is applied to the surface. If **TWALL**=0.0, an adiabatic wall is assumed. For **TWALL**>0.0, a constant temperature surface with a nondimensional wall temperature of **TWALL** defined as:

$$(T_{wall})_{non-dimensional} = \frac{T_{wall}}{T_{ref}}$$

is imposed. (Here T_{ref} is the reference temperature imposed by the input file variable **TREF**. A value of **TWALL**<0.0 is not permitted. Naturally, poor convergence or solution divergence can occur if **RPMWALL** or **TWALL** suggest boundary values which are significantly different from the remainder of the flowfield. In such cases where this occurs, it is recommended that the solution be started with more conservative boundary values, and then restarted using the final boundary values.

Restrictions/Limitations

The **SSVI** boundary specification is restricted to 3-D mesh surfaces (2-D mesh surfaces should use the **SS2DVI** boundary specification). The boundary rotational speed imposed by the **SSVI** boundary condition can only be nonzero when using the cylindrical coordinate solution algorithm in the *ADPAC08* code. When using the Cartesian coordinate solution algorithm **FCART,FCARB**= 1.0, the boundary rotational speed must be zero (**RPMWALL**= 0.0 when **FCART,FCARB**= 1.0). Refer to the Chapter on input file parameters for a description of **TREF**, **RPM**, **FCARB**, and **FCART**.

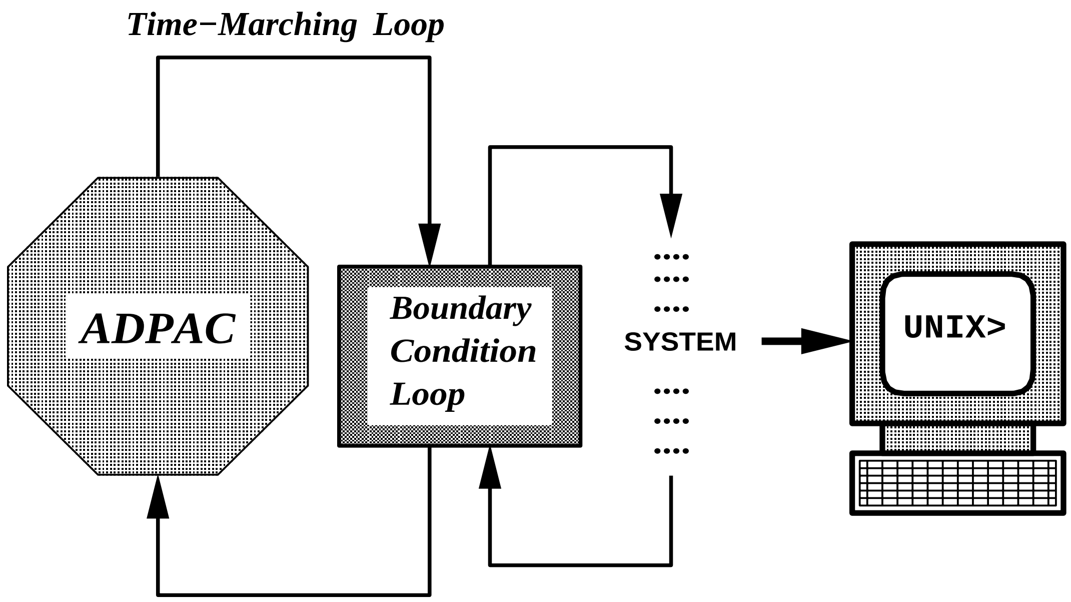
Common Errors

- Incorrect sign for value of boundary rotational speed **RPMWALL**.

- Attempt to utilize a nonzero boundary rotational speed with the Cartesian coordinate solution algorithm.
- Application of **SSVI** to a 2-D mesh system.

SYSTEM

ADPAC08 UNIX system Call Specification



Application

The **SYSTEM** specification is not a boundary condition as such, but directs the *ADPAC08* code to perform a UNIX system call at every application of the boundary condition loop. In the application illustrated above, every time the *ADPAC08* code encounters the boundary condition specification **SYSTEM** the code is directed to perform a UNIX system call of the command *UPDATEBC*, which is presumably a user-specified code used to update certain boundary variables (see sample format below). This new data could then be imported using the **BDATIN** boundary specification. The **SYSTEM** function can quickly lead to trouble due to the number of times it is called within the time-marching strategy, and the user should thoroughly review the documentation below before attempting to use this facility.

Boundary Data File Format

The boundary data file specifications for the mesh interface indicated in the illustrative graphic for the **SYSTEM** boundary condition are given below:

```
SYSTEM 1 1 J J P P I K   1   1  11  21  1 11  11  21  1 11
INTERVAL
1
COMMAND
/usr/local/bin/updatebc
```

Note that a complete **SYSTEM** specification requires the specification of additional data beyond the standard boundary specification line.

Description

The **SYSTEM** statement is provided to permit the specification of a UNIX system call from within the *ADPAC08* code. Once the **SYSTEM** specification is directed into the *ADPAC08* code, at a specified interval of iterations, during every execution of the boundary condition loop, when the **SYSTEM** specification is encountered, the code executes the command provided by the **SYSTEM** specification and pending successful completion, continues execution. The **SYSTEM** specification is based on the FORTRAN instrinsic *system* function which must be available in the compiling system. It should be noted that the command dictated by the **SYSTEM** specification will be performed *every* time the boundary condition loop is encountered. This suggests that the system call will be made a minimum of four times for each iteration of the time-marching scheme (for the four stage scheme). This number can grow rapidly if more complicated iteration strategies are used such as multigrid, subiterations, etc., and the user should be warned that such redundant system calls can wreak havoc on an otherwise friendly solution. A **SYSTEM** specification, in conjunction with the **BDATIN/BDATOU** boundary data specifiers can be effectively combined to provide a run time interface between the *ADPAC08* code and an external flow solver.

A **SYSTEM** specification requires four additional lines in addition to the normal boundary data file descriptor, as shown above. The first additional line simply contains the label for the iteration interval **INTERVAL**, followed by the actual value of **INTERVAL** The **SYSTEM** routine will only be invoked every **INTERVAL** time-marching iterations. The next line contains the label for the system call command (**COMMAND**) variable. The following line contains the actual UNIX command to be issued at every **SYSTEM** encounter in the boundary condition loop.

Restrictions/Limitations

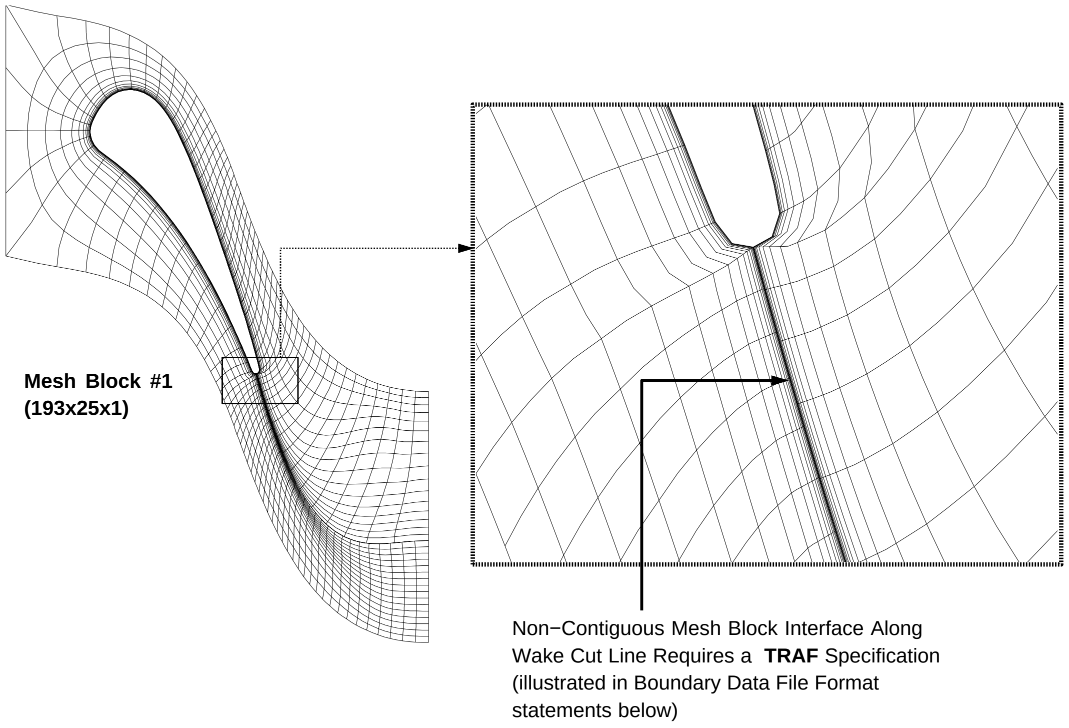
Data provided in the **SYSTEM** specification should represent a valid UNIX system command. The FORTRAN intrinsic function *system* must be available on the compiling system. The **SYSTEM** function is not available on the Cray or nCUBE computers.

Common Errors

- Invalid UNIX system command provided in **SYSTEM** boundary specification.
- Failure to provide the additional data **INTERVAL** and **COMMAND** for **SYSTEM** specification.
- FORTRAN intrinsic function *system* unavailable at compile time.
- User unaware that **SYSTEM** action occurs four or more times per iteration.

TRAF

TRAF2D/3D Type Non-Contiguous Mesh Block Interface Patching Scheme


Mesh Block #1
(193x25x1)
Non−Contiguous Mesh Block Interface Along Wake Cut Line Requires a **TRAF** Specification (illustrated in Boundary Data File Format statements below)

Application

The **TRAF** specification was developed specifically to treat C-type mesh systems for airfoil cascades with a noncontiguous wake cut line such as those developed using the *TRAF2D/TRAF3D* [16] flow solver. The example graphic above illustrates a two-dimensional mesh system used to predict the flow through a turbine vane passage. This mesh was generated using the *JERRYC/TOMC* mesh generation package which was developed for the *TRAF2D/TRAF3D* flow solver. The C-type mesh utilizes a noncontiguous wake cut line as shown in the trailing edge detail. The **TRAF** specification is applied along either side of the wake cut line to permit communication of flow variables across the noncontiguous mesh interface.

Boundary Data File Format

The boundary data file specifications for the mesh interface indicated in the illustrative graphic for the **TRAF** boundary condition are given below:

```
TRAF     1  1  J  J  P  P  L  L   1   1   1  33   1   2 177  193   1   2
TRAF     1  1  J  J  P  P  L  L   1   1 177 193   1   2   1   33   1   2
```

Note that a complete **TRAF** specification generally requires two **TRAF** statement lines in the boundary data file. In the example above, the first specification provides the interblock communication for one side of the C-grid wake cut, while the second specification provides the communication for the other side of the C-grid wake cut. It is a common error to underspecify a **TRAF** boundary by only providing a single line per interface.

Description

The **TRAF** boundary statement provides a means for block to block communication for cases involving neighboring mesh boundaries which share a common surface, but are non contiguous in one grid index. The standard example of this type of mesh is a C-type mesh about an airfoil where the points along the C-grid wake cut line are noncontiguous. This type of mesh system has been utilized extensively in the *TRAF2D/TRAF3D* [16] flow solver system, and the **TRAF** boundary specification has been provided to permit *ADPAC08* execution on these meshes. The implied restriction of the **TRAF** boundary specification is that the mesh is only misaligned in one coordinate direction, specifically the i coordinate. It is also assumed that the endpoints of the **TRAF** boundary specification are contiguous. As such, the **TRAF** boundary specification is fairly restrictive, and should not be used as a general purpose misaligned mesh routine. An example of an appropriate application of the **TRAF** specification is given in the illustrative graphic. The **TRAF** boundary specification is valid for either 2-D or 3-D mesh blocks. For 2-D mesh blocks, the **TRAF** specification *must* be applied to a j=constant boundary. For 3-D mesh blocks, the **TRAF** specification *must* be applied to a k=constant boundary. The **TRAF** boundary condition requires no additional data beyond the initial specification line, but does require some extra care when used. The primary precaution is that the **TRAF** procedure is based entirely on a simplified interpolations scheme, and hence, does not maintain either global or local conservation of flow variables across the mesh interface.

Restrictions/Limitations

The **TRAF** boundary specification is restricted to mesh interfaces which lie on a common surface (no significant overlap). The **TRAF** procedure permits only that the i coordinates between adjacent mesh surfaces are misaligned. The **TRAF** procedure is only valid if the misaligned i coordinates either increase or decrease in the x direction monotonically. The endpoints of the **TRAF** specification surface must ne contiguous. The **TRAF** specification may only be applied to j=constant surfaces for 2-D mesh blocks, and k=constant surfaces for 3-D mesh blocks.

Common Errors

- Failure to provide 2 **TRAF** statements for each interface.
- Incorrectly specified or misaligned extents of boundary regions (values of **M1LIM1, M1LIM2, N1LIM1, N1LIM2, M2LIM1, M2LIM2, N2LIM1, N2LIM2** do not correctly define the interface extents on blocks **LBLOCK1** and **LBLOCK2**).
- Attempt to use **TRAF** for a boundary which has 2 misaligned coordinates.

- Attempt to use **TRAF** for boundaries which are not monotonic in the x direction.
- Application of **TRAF** to mesh interfaces which do not have contiguous end points.
- Application of **TRAF** to an i=constant or k=constant mesh surface in a 2-D block.
- Application of **TRAF** to an i=constant or j=constant mesh surface in a 3-D block.

3.8 Mesh File Description

The *ADPAC08 case*.**mesh** file is a data file containing the x, y, z grid coordinates of the multiple mesh blocks which are read in to define the physical grid points used in the time-marching solution (see Section 3.5 for a description of the *case* name and the mesh file naming convention). The mesh coordinates are specified in a Cartesian frame of reference, as shown in Figure 3.10, although the *ADPAC08* program may ultimately convert these coordinates to a cylindrical coordinate system during execution. The mesh coordinates are stored in what is known as *PLOT3D* multiple grid format, and are formatted using the Scientific Database Library (SDBLIB). (The SDBLIB system allows machine-independent binary file storage.) The *case*.**mesh** file **must** be available for every *ADPAC08* run. At the beginning of program execution, the *ADPAC08* program attempts to open the mesh file and read in the mesh size to make sure that enough memory has been allocated for the given problem. If the mesh file is not found, or if the mesh is too large, the appropriate error message is issued, and the program will terminate.

Mesh coordinates are **assumed** to be nondimensional numbers. The *ADPAC08* employs a dimensional scaling factor (see input file variable **DIAM**) to convert the nondimensional mesh coordinates into dimensional coordinates with units of feet. If the mesh is generated with units of feet, then the dimensionalizing factor is simply 1.0. Proper nondimensionalization and specification of the dimensionalizing factor **DIAM** is required in order to accurately achieve the desired flow Reynolds number and rotational speed (see the discussion of input variable **ADVR** is Section 3.6). It is also required that the ordering of the mesh points form a "left-handed" mesh. This implies that at every point in the mesh, the vectors representing the positive i, j, and k coordinate directions form a left-handed coordinate system (see Figure 3.11). Consider the case of a sheared H-grid discretizing a single blade passage of a compressor (this type of mesh is used extensively in the Standard Configurations described in Chapter 5). If we assume that looking downstream through the blade passage is essentially the positive i direction, and that the radial direction from hub to tip is essentially the positive j direction, a left-handed mesh would require that the positive k direction be from right to left in this orientation.

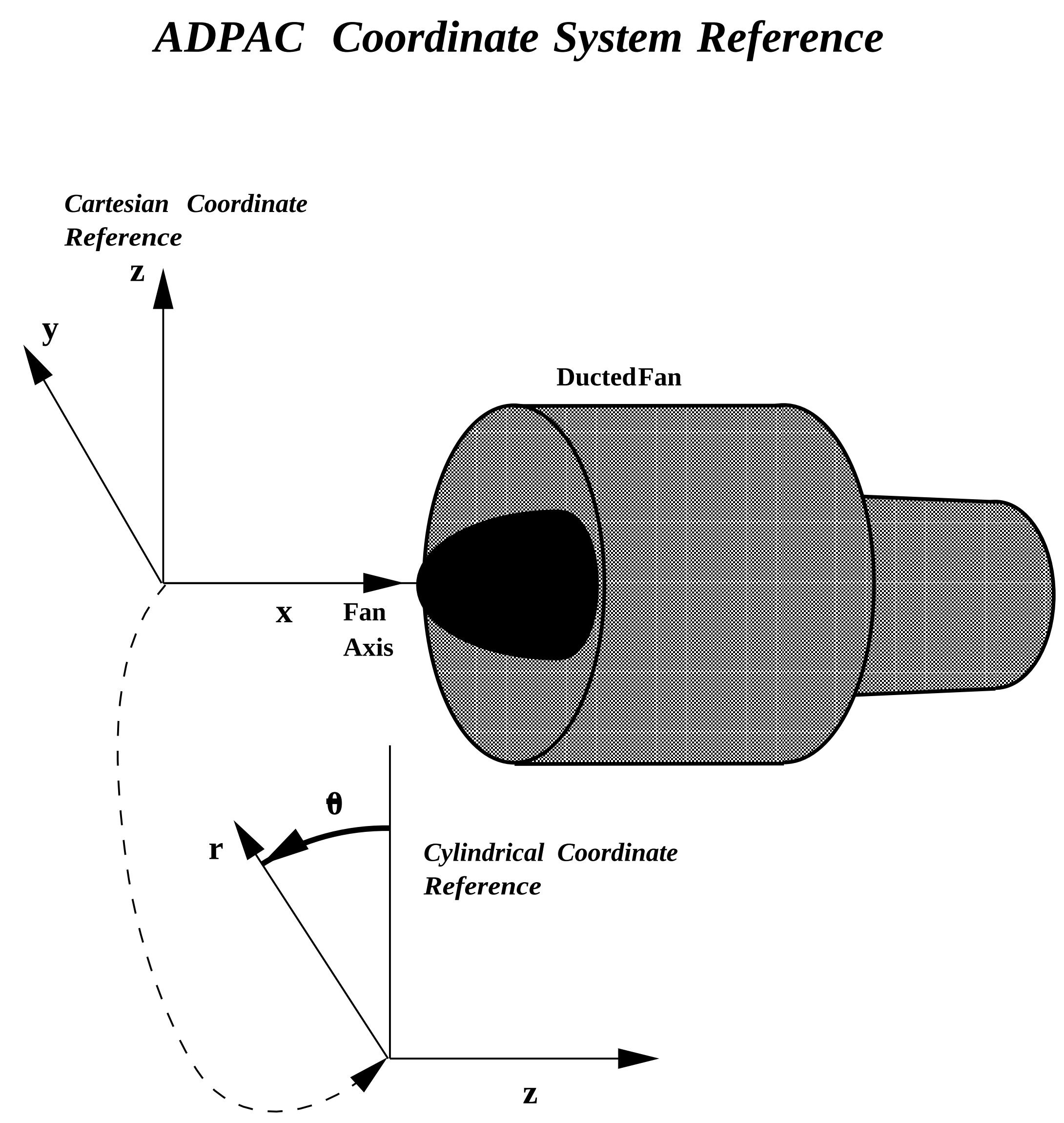

ADPAC Coordinate System Reference
Cartesian Coordinate Reference
z
y
x
Fan Axis
DuctedFan
θ
r
Cylindrical Coordinate Reference
z

Figure 3.10: *ADPAC08* Mesh Coordinate Reference Description

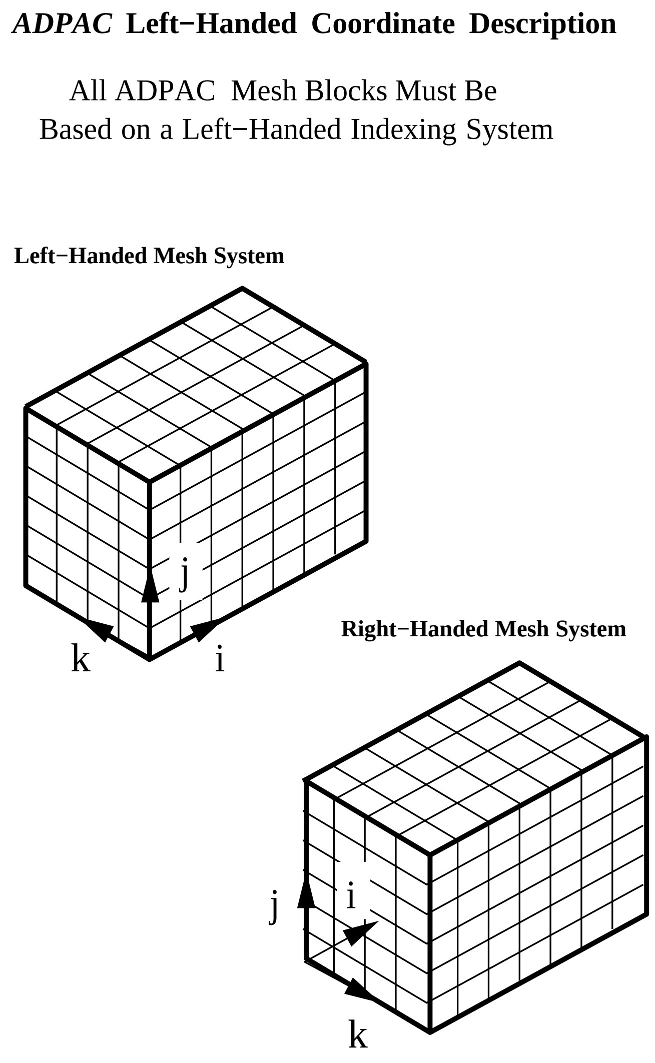

ADPAC Left−Handed Coordinate Description
All ADPAC Mesh Blocks Must Be Based on a Left−Handed Indexing System
Left−Handed Mesh System
j
k
i
Right−Handed Mesh System
j
i
k

Figure 3.11: *ADPAC08* Left-Handed Coordinate System Description

In order to understand the *PLOT3D* multiple-grid mesh file format, and the utilization of the *SDBLIB* routines, a comparison of the FORTRAN coding for each method is given below for comparison.

The FORTRAN coding to read a *PLOT3D* unformatted multiple-block mesh file might be given as:

PLOT3D Mesh File Format FORTRAN Coding Example

```
      OPEN(UNIT=IGRID,FILE=FNAME,FORM='UNFORMATTED',STATUS='OLD')
      READ(IGRID) MG
      READ(IGRID) (IL(L), JL(L), KL(L),L=1,MG)
      DO 10 L = 1, MG
      READ(IGRID) (((X(I,J,K,L),I=1,IL(L)),J=1,JL(L)),K=1,KL(L)),
     .            (((Y(I,J,K,L),I=1,IL(L)),J=1,JL(L)),K=1,KL(L)),
     .            (((Z(I,J,K,L),I=1,IL(L)),J=1,JL(L)),K=1,KL(L))
10    CONTINUE
```

Each of the terms used in the FORTRAN code given above are defined below:

IGRID	FORTRAN unit number for read statement
FNAME	File name for mesh file
MG	number of grid blocks
IL(L)	maximum *i* grid index for block L
JL(L)	maximum *j* grid index for block L
KL(L)	maximum *k* grid index for block L
X(I,J,K,L)	Cartesian coordinate value of x for point (I,J,K) in block L
Y(I,J,K,L)	Cartesian coordinate value of y for point (I,J,K) in block L
Z(I,J,K,L)	Cartesian coordinate value of z for point (I,J,K) in block L

An example of the corresponding FORTRAN coding to read an *ADPAC08* binary mesh file using the Scientific Database Library (SDBLIB) routines is given below:

PLOT3D Mesh File Format FORTRAN Coding Example Using SDBLIB

```
       CALL QDOPEN( IGRID, FNAME, JE )
       CALL QDGETI( IGRID, MG   , JE )
       DO L = 1, MG
       CALL QDGETI( IGRID, IL(L), JE )
       CALL QDGETI( IGRID, JL(L), JE )
       CALL QDGETI( IGRID, KL(L), JE )
       ENDDO
       IPOINT = 1
       DO 10 L = 1, MG
       ILENGTH = IL(L) * JL(L) * KL(L)
       CALL QDGEEA( IGRID, X(IPOINT), ILENGTH, JE )
       CALL QDGEEA( IGRID, Y(IPOINT), ILENGTH, JE )
       CALL QDGEEA( IGRID, Z(IPOINT), ILENGTH, JE )
       IPOINT = IPOINT + ILENGTH
10     CONTINUE
       CALL QDCLOS( IGRID, JE )
```

A listing of the additional terms used in the coding above is given below:

QDOPEN	SDBLIB routine to open a file for input or output
QDGETI	SDBLIB routine to get an integer
QDGEIA	SDBLIB routine to get an integer array of length ILENGTH
QDGETE	SDBLIB routine to get a real number
QDGEEA	SDBLIB routine to get a real array of length ILENGTH
QDCLOS	SDBLIB routine to close a file

IGRID	FORTRAN logical unit number for grid input
JE	An error trigger; 0 for no error, 1 if an error occurs
IB	Integer array containing the IL, JL, and KL grid block sizes
ILENGTH	Integer length of an array of data
IPOINT	Integer pointer for block L to locate the initial memory location for a block of data

The x, y, z coordinates are read in as a single-dimensioned array in the *SDBLIB* format, and the *ADPAC08* program includes a conversion routine (source file *convas.f*) which converts the single dimension array data to a three-dimensional data array.

The mesh file may be utilized directly with the *PLOT3D* program when the default real number size of the compiled *PLOT3D* code is defined as 32 bits (as it is on many workstations). The corresponding *PLOT3D* read command for an *ADPAC08* mesh file are:

```
PLOT3D PROMPT> read/mg/bin/x=case.mesh
```

Obviously the user should substitute their own case name in the *PLOT3D* input line.

Unformatted mesh files may be converted to *ADPAC08* format using the *MAKEADGRID* program described in Chapter 7.

3.9 Body Force File Description

The *ADPAC08* body force file is a data file containing the blade blockage, body force, and energy source terms used in a 2-D axisymmetric representation of an embedded blade row (see 2-D/3-D Solution Concepts, Section 2.3). Individual body force files contain the cell-centered blade blockage, body forces, and energy source terms for a specific mesh block. As a result, the file naming procedure for the body force file is somewhat different than the mesh, plot and restart files, where a single file contains all the data for a multiple-block solution (a complete description of the *ADPAC08* file naming procedure is given in Section 3.5).

The terms in the body force file are stored in binary format, based on the Scientific Database Library routines. (The SDBLIB system permits machine-independent binary file storage.) The blockage, body forces, and energy sources are stored as nondimensional numbers using the nondimensionalization strategy listed in Section 1.2 of the Final Report [1].

In order to understand the body force file format, and the utilization of the *SDBLIB* routines, a representative FORTRAN coding example to read in a body force file is given below for comparison.

Body Force File Format FORTRAN Coding Example Using SDBLIB

```
      CALL QDOPEN( IBODY, FNAME, JE )
      ILENGTH = 3
      CALL QDGEIA( IBODY, IB, ILENGTH, JE )
      IMX = IB(1)
      JMX = IB(2)
      KMX = IB(3)
      ILENGTH = IMX * JMX * KMX
      CALL QDGETE( IBODY, DUMMY, JE )
      CALL QDGETE( IBODY, DUMMY, JE )
      CALL QDGETE( IBODY, DUMMY, JE )
      CALL QDGETE( IBODY, DUMMY, JE )
      CALL QDGEEA( IBODY, BFR  (IPOINT(L)), ILENGTH, JE )
      CALL QDGEEA( IBODY, BFRU (IPOINT(L)), ILENGTH, JE )
      CALL QDGEEA( IBODY, BFRV (IPOINT(L)), ILENGTH, JE )
      CALL QDGEEA( IBODY, BFRW (IPOINT(L)), ILENGTH, JE )
      CALL QDGEEA( IBODY, BFRE (IPOINT(L)), ILENGTH, JE )
      CALL QDGEEA( IBODY, BL   (IPOINT(L)), ILENGTH, JE )
      CALL QDCLOS( IBODY, JE )
```

A listing of the FORTRAN variables and their meanings is given below:

QDOPEN	SDBLIB routine to open a file for input or output
QDGETI	SDBLIB routine to get an integer
QDGETE	SDBLIB routine to get a real number
QDGEIA	SDBLIB routine to get an integer array of length ILENGTH
QDGEEA	SDBLIB routine to get a real array of length ILENGTH
QDCLOS	SDBLIB routine to close a file
IBODY	FORTRAN logical unit number for body force file input
JE	An error trigger; 0 for no error, 1 if an error occurs
IB	Integer array containing the IL, JL, and KL grid block sizes
IMX	Mesh size+1 in the i coordinate direction
JMX	Mesh size+1 in the j coordinate direction
KMX	Mesh size+1 in the k coordinate direction
ILENGTH	Integer length of an array of data
IPOINT(L)	Integer pointer for block L to locate the initial memory location for a block of data
BFR	Body force for density (continuity equation)
BFRU	Body force for axial momentum
BFRV	Body force for radial momentum
BFRW	Body force for circumferential momentum
BFRE	Body force for internal energy
BL	Blockage term

The body force data are read and written as a single-dimensioned array in the *SDBLIB* format, and the *ADPAC08* program includes a conversion routine (source file *convas.f*) which converts the three-dimensional array data to the single dimension array data.

3.10 Standard Output File Description

The *ADPAC08* standard output file *case*.**output** provides information regarding the status of a particular calculation during code execution. The status information includes startup, code response to input files (mesh, restart, standard input, and boundary data), convergence history, error messages, and output file generation (plot files, restart files, body force files). The information given in the standard output file is essentially self explanatory, so no further description is given here. A sample output file is included in the standard distribution of the *ADPAC08* code for the test case described in Appendix A. Additional details may be found in this Appendix.

3.11 Plot File Description

The *ADPAC08* *case*.**p3dabs** and *case*.**p3drel** plot files contain predicted absolute and relative flow data values, respectively, for each of the mesh points in a multiple-block mesh *ADPAC08* solution. The grid-centered aerodynamic data is obtained by algebraically averaging the cell-centered data generated by the finite-volume solver during the time-marching process. As a result of the averaging procedure, this data can occasionally appear inconsistent at the corners of a mesh block, and should therefore only be used for graphical viewing, and not for post processing to obtain performance data, mass flow rates, pressure rise, etc. The flow plot data are specified in a Cartesian coordinate system (velocities are u_x, u_y, u_z) to be consistent with the representation of the mesh file (see Section 3.8). The plot files are written in what is known as *PLOT3D* multiple grid binary format. The plot data are formatted using the Scientific Database Library (SDBLIB). (The SDBLIB system permits machine-independent binary file storage.) The flow data are listed as nondimensional numbers using the nondimensionalization strategy listed in Section 1.2 of the Final Report [1].

In order to understand the *PLOT3D* multiple-grid flow file format, and the utilization of the *SDBLIB* routines, a comparison of the FORTRAN coding for each method is given below for comparison.

The equivalent FORTRAN coding for an unformatted *PLOT3D* flow file could be given as:

PLOT3D Flow File Format FORTRAN Coding Example

```
      WRITE( ) MG
      WRITE( ) (IL(L), JL(L), KL(L),L=1,MG)
      DO 20 L = 1, MG
      WRITE( ) EM(L), REY(L), ALF(L), TIME(L)
      WRITE( ) (((R (I,J,K,L),I=1,IL(L)),J=1,JL(L)),K=1,KL(L)),
     .         (((RU(I,J,K,L),I=1,IL(L)),J=1,JL(L)),K=1,KL(L)),
     .         (((RV(I,J,K,L),I=1,IL(L)),J=1,JL(L)),K=1,KL(L)),
     .         (((RW(I,J,K,L),I=1,IL(L)),J=1,JL(L)),K=1,KL(L)),
     .         (((RE(I,J,K,L),I=1,IL(L)),J=1,JL(L)),K=1,KL(L))
20    CONTINUE
```

Each of the terms used in the FORTRAN code given above are defined below:

MG	number of grid blocks
IL(L)	maximum i grid index for block L
JL(L)	maximum j grid index for block L
KL(L)	maximum k grid index for block L
X(I,J,K,L)	Cartesian coordinate value of x for point (I,J,K) in block L
Y(I,J,K,L)	Cartesian coordinate value of y for point (I,J,K) in block L
Z(I,J,K,L)	Cartesian coordinate value of z for point (I,J,K) in block L
EM(L)	*PLOT3D* Reference Mach number for block L
REY(L)	*PLOT3D* Reference Reynolds number for block L
ALF(L)	*PLOT3D* Reference angle for block L

TIME(L)	*PLOT3D* Reference time for block L
R (I,J,K,L)	ρ at point (I,J,K) in block L
RU(I,J,K,L)	ρu_x at point (I,J,K) in block L
RV(I,J,K,L)	ρu_y at point (I,J,K) in block L
RW(I,J,K,L)	ρu_z at point (I,J,K) in block L
RE(I,J,K,L)	ρe at point (I,J,K) in block L

<u>PLOT3D Flow File Format FORTRAN Coding Example Using SDBLIB</u>

```
      CALL QDOPEN( IFLOW, FNAME, JE )
      CALL QDGETI( IFLOW, MG   , JE )
      DO L = 1, MG
      CALL QDGETI( IFLOW, IL(L), JE )
      CALL QDGETI( IFLOW, JL(L), JE )
      CALL QDGETI( IFLOW, KL(L), JE )
      ENDDO
      IPOINT = 1
      DO 20 L = 1, MG
      ILENGTH = IL(L) * JL(L) * KL(L)
      CALL QDGETE( IFLOW, EM(L)  , JE )
      CALL QDGETE( IFLOW, REY(L) , JE )
      CALL QDGETE( IFLOW, ALF(L) , JE )
      CALL QDGETE( IFLOW, TIME(L), JE )
      CALL QDGEEA( IFLOW, R (IPOINT), ILENGTH, JE )
      CALL QDGEEA( IFLOW, RU(IPOINT), ILENGTH, JE )
      CALL QDGEEA( IFLOW, RV(IPOINT), ILENGTH, JE )
      CALL QDGEEA( IFLOW, RW(IPOINT), ILENGTH, JE )
      CALL QDGEEA( IFLOW, RE(IPOINT), ILENGTH, JE )
      IPOINT = IPOINT + ILENGTH
20    CONTINUE
      CALL QDCLOS( IFLOW, JE )
```

A listing of the additional terms used in the coding above is given below:

QDOPEN	SDBLIB routine to open a file for input or output
QDGETI	SDBLIB routine to get an integer
QDGEIA	SDBLIB routine to get an integer array of length ILENGTH
QDGETE	SDBLIB routine to get a real number
QDGEEA	SDBLIB routine to get a real array of length ILENGTH
QDCLOS	SDBLIB routine to close a file
IFLOW	FORTRAN logical unit number for flow input
JE	An error trigger; 0 for no error, 1 if an error occurs
IB	Integer array containing the IL, JL, and KL grid block sizes
ILENGTH	Integer length of an array of data
IPOINT	Integer pointer for block L to locate the initial memory location for a block of data

The flow data are written as a single-dimensioned array in the *SDBLIB* format, and the *ADPAC08* program includes a conversion routine (source file *convas.f*) which converts the three-dimensional array data to the single dimension array data.

The plot files may be utilized directly with the *PLOT3D* program when the default real number size of the compiled *PLOT3D* code is defined as 32 bits (as it is on many workstations). The corresponding *PLOT3D* read commands for an *ADPAC08* mesh and flow file are:

```
PLOT3D PROMPT> read/mg/bin/x=case.mesh/q=case.p3dabs
```

Obviously the user should substitute their own case name in the *PLOT3D* input line.

3.12 Restart File Description

The *ADPAC08* restart file is a data file containing the cell-centered flow variables generated during an *ADPAC08* solution. This file is intended to permit continued execution of the code from the point at which a previous calculation was terminated. This feature permits breaking large jobs into smaller computational pieces. This process of job restarting is considered a good practice to avoid loss of data due to computer malfunctions and job quota limitations. At the end of a given job, whether the calculation is a restart run or not, the *ADPAC08* program will attempt to write out the current cell centered data to the file *case.restart.new* (see Section 3.2 for a description of the file naming convention). The restart file may then be used to continue the calculation at this same point by simply renaming the file *case.restart.new* to *case.restart.old*, setting the input trigger appropriately (see the description of **FREST** in Section 3.6), and rerunning the code. The restart data are written in either the cylindrical or Cartesian coordinate system depending on the variable format used during execution of the *ADPAC08* code for each particular mesh block. Velocities are specified as u_z, u_r, u_θ, and all flow variables utilize the nondimensionalization strategy described in Section 1.2 of the companion Final Report [1].

In order to demonstrate the format of the restart file, a sample of the FORTRAN coding utilizing the *SDBLIB* library required to read a restart file is given below.

***ADPAC08* Restart Flow File Format FORTRAN Coding Example**

```
      CALL QDOPEN( IREST, FNAME, JE )
      CALL QDGETI( IREST, MG   , JE )
      NLENGTH = 3 * MG
      CALL QDGEIA( IREST, IB, NLENGTH, JE )
        DO 10 N = 1, MG
        IMX(N) = IB((N-1)*3+1)
        JMX(N) = IB((N-1)*3+2)
        KMX(N) = IB((N-1)*3+3)
        LENGTH = IMX(N) * JMX(N) * KMX(N)
        CALL QDGEEA( IREST, R (IJK(N)), LENGTH, JE )
        CALL QDGEEA( IREST, RU(IJK(N)), LENGTH, JE )
        CALL QDGEEA( IREST, RV(IJK(N)), LENGTH, JE )
        CALL QDGEEA( IREST, RW(IJK(N)), LENGTH, JE )
        CALL QDGEEA( IREST, RE(IJK(N)), LENGTH, JE )
        CALL QDGEEA( IREST, P (IJK(N)), LENGTH, JE )
49      CONTINUE
       NLENGTH = MG
       CALL QDGEIA( IREST, NCYC    , NLENGTH  , JE )
       CALL QDGEIA( IREST, DTHETA  , NLENGTH  , JE )
       CALL QDGEIA( IREST, OMEGAL  , NLENGTH  , JE )
       CALL QDGEIA( IREST, JE )
```

Each of the terms used in the FORTRAN code given above are defined below:

MG	number of grid blocks
IMX(L)	maximum i grid index for block L
JMX(L)	maximum j grid index for block L
KMX(L)	maximum k grid index for block L
R (IJK(L))	ρ at point IJK(L) in block L
RU(IJK(L))	ρu_x at point IJK(L) in block L
RV(IJK(L))	ρu_y at point IJK(L) in block L
RW(IJK(L))	ρu_z at point IJK(L) in block L
RE(IJK(L))	ρe at point IJK(L) in block L
P (IJK(L))	pressure at point IJK(L) in block L
QDOPEN	SDBLIB routine to open a file for input or output
QDGETI	SDBLIB routine to get an integer
QDGEIA	SDBLIB routine to get an integer array of length ILENGTH
QDGEEA	SDBLIB routine to get a real array of length ILENGTH
QDCLOS	SDBLIB routine to close a file

IREST	FORTRAN logical unit number for restart input
JE	An error trigger; 0 for no error, 1 if an error occurs
IB	Integer array containing the IMAX, JMAX, and KMAX grid block sizes
ILENGTH	Integer length of an array of data
IJK(L)	Integer pointer for block L to locate the initial memory location for a block of data

The restart data are written as a single-dimensioned array in the *SDBLIB* format, and the *ADPAC08* program includes a conversion routine (source file *convas.f*) which converts the three-dimensional array data to single dimension array data.

3.13 Convergence File Description

The *ADPAC08* convergence history file *case.***converge** (see Section 3.5 for a description of the *ADPAC08* file naming convention) is an ASCII data file which contains the residual convergence history of the time-marching solution. The residual history is useful for determining whether the numerical solution has converged sufficiently to permit interrogation of the numerical results, or whether additional restarted calculations are required to obtain an accurate solution. Typically, a solution is deemed converged when the residuals have been reduced by three orders of magnitude or more. The data in the *case.***converge** file are organized in the following format:

```
iteration        log10(maximum )    log10(root-mean square)
number               (residual)         (residual        )
   .                    .                    .
   .                    .                    .
   .                    .                    .
   .                    .                    .
```

The residual R at any cell in the finite volume solution is calculated as the sum of the changes in the 5 conservation variables ρ, ρu, ρv, ρw, and ρe.

The maximum residual is then defined as the maximum of all the residuals over all the cells of all mesh blocks. The root-mean square residual is the square root of the sum of the squares of all the cells for all mesh blocks. The *case.***converge** file residual data are reported as the base 10 logarithm of the actual residuals in order to quickly evaluate the convergence of the solution (if the reported log10 maximum residual starts at -2.5 and ends up at -5.5, the solution has converged three orders of magnitude). The *case.***converge** file is formatted in columns to permit convenient plotting using any of a number of x-y plotting programs (the *FULLPLOT* program described in Reference [3] is one example).

3.14 Image File Description

The *ADPAC08* graphics display system (see Chapter 9) has the capability of saving a raster image of the local graphics screen to a file at specific iteration intervals using the Silicon Graphics image file format. This feature is included as a simple means of constructing flowfield animations. The input variables dealing with this facility **FGRAFIX, FGRAFINT, FIMGSAV, FIMGINT** are described in Section 3.6, and the image file naming convention is described in Section 3.5. In short, image files can be saved when the graphics display system is running on a single Silicon Graphics workstation or across a network between two Silicon Graphics workstations supporting the IRIX Operating System Version 4.0.1 or above, and also supporting the IRIX *scrsave* command. Image files can be viewed after they have been saved by issuing the command

ipaste *case.img.#*

Other IRIS-specific commands such as *imgview, movie* and others may also be suitable for viewing image files. Additional information on the IRIS image format and the image manipulation commands are available in the Silicon Graphics system documentation.

3.15 Troubleshooting an *ADPAC08* Failure

The *ADPAC08* code contain a large number of error checking and handling facilities to determine and report to the user when a problem in the calculation occurs. Unfortunately, some problems simply cannot be detected and it may occur that for a particular case the solution will diverge (uncontrolled increase in solution residual) or simply "blow up" as a result of numerical difficulties or an invalid numerical operation (divide by zero for example). These cases are notoriously frustrating for the user because the cause is often difficult to identify. The paragraphs below attempt to provide a structured approach to rectifying numerical problems for an *ADPAC08* run based on the author's experience.

Step 1.) Carefully Check the *ADPAC08* Input File for Errors

The *ADPAC08* standard input file controls the overall characteristics of the computational process, and as such, plays a large role in determining the behavior of a job. Typical parameters to check are to make sure that the **CFL** variable is negative for steady state calculations, and positive for time-accurate calculations, and to make sure that the absolute value is not too large (5.0 is a typical magnitude). If the **CFL** value is greater than the **CFMAX** variable, or generally if the magnitude of **CFL** is larger than 2.5, then residual smoothing *must* be activated (**FRESID**=1.0). Naturally, the values for **VIS2** and **VIS4** should also be within the limits suggested for those values. A common problem for rotating geometries is an incorrect rotational speed, or simply the wrong sign on the rotational speed (rotating the wrong way), so check the values of **RPM** and/or **ADVR** carefully. The user can also selectively turn off features such as the turbulence model (see **FTURBB**) and/or multigrid (see **FMULTI**) to check on their influence on the stability of the solution. Finally, the user should make sure that the proper **CASENAME** is specified in the input file. Other problems are discussed in the individual input file variable descriptions in Section 3.6.

Step 2.) Carefully Check the *ADPAC08* Boundary Data File for Errors

The *ADPAC08* boundary data file controls the application of boundary conditions on the various mesh surfaces necessary to define the flow characteristics of an *ADPAC08* run. Common errors in the boundary data file include mismatched **PATCH** specifications, incorrectly specifying inflow data (particularly when **INLETT** is used), and incorrectly specifying rotational speeds for solid surfaces using **SSVI**. If the solution will run for a few iterations, it may be helpful to get a PLOT3D output file at this point and examine the solution using **PLOT3D** or **FAST**. Check for obvious solution features such as flow going the right direction, contour lines matching at **PATCH** boundaries (although contour lines may not match exactly at any mesh corner), and obvious radical changes in flow variables (total pressures and/or total temperatures which are very large or negative). The user can often trace a faulty boundary condition by selectively commenting several specifications from the boundary data file and rerunning to see if the same problem occurs. If the solution diverges even when no boundary conditions are specified, then a problem exists in the mesh or input file. Other boundary condition specific common errors are discussed in the individual boundary data file variable descriptions in Section 3.7.

Step 3.) Carefully Check the *ADPAC08* Mesh File for Errors

Most common problems encountered when the *ADPAC08* code does not perform adequately can be traced to poor mesh quality. Although the mesh may be free from obvious flaws such as crossed mesh lines and/or zero volumes, this does not guarantee that numerical difficulties will be avoided. The most common overlooked features of mesh quality are the mesh expansion ratio and the mesh shear angle. Mesh expansion ratio relates to the change in physical mesh spacing along a given coordinate direction from one point to another. For stability, the maximum mesh expansion ratio at any point should not exceed 1.3. The *ADPAC08* provides a listing of maximum mesh expansion ratios for each grid block and issues a warning if the mesh expansion ratio exceeds 1.3. The code can tolerate larger ratios in many cases, but definite problems can be expected if the maximum expansion ratio gets larger than 2.0. Mesh shear can also cause problems. The more orthogonal the mesh, the less likely mesh-induced numerical difficulties will occur. Another potential mesh problem involves mesh cells with very small radii (such as along a sting upstream of a propeller, etc.) which may require increasing the diameter of the sting to prevent problems. Application of the multigrid iteration strategy and reducing the value of **EPSY** in the input file have been found to be effective remedies for such problems.

Step 4.) Check for the Possibility of an Invalid Flow Condition

The author's experience has been that many users feel if a problem can be defined then it should possess a solution. In fluid dynamics this is certainly not true. If a solution is attempted for a fan rotor, for example, at a pressure ratio which is beyond the stall point for that rotor, then no solution exists and the code will very likely diverge without explanation. In many cases, the numerical equivalent of an invalid flow condition is that the solution will either not converge, or will simply diverge. Another common example is attempting to extract a steady state solution for a problem which is truly time-dependent. Blunt body flows often result in a time-dependent solution due to vortex shedding, and the steady state analysis of this flow will likely never converge. This behavior also occurs frequently when a strong adverse pressure gradient or flow separation is present in the solution. Now it is true that in some cases, the level of convergence may also be limited by such factors as mesh quality, numerical accuracy, and/or turbulence model limit cycles, and it is difficult to determine whether the cause is numerical or physical. This is unfortunately a matter of experience and the user is encouraged to question whether their case can truly have a "steady" solution.

Step 5.) Determine if the Problem is Computer Dependent

The *ADPAC08* code was developed and tested on UNIX-based operating systems using FORTRAN 77 standard coding techniques. In spite of the standardization in the computer industry, not all machines produce the same answer for a given problem due to compiler optimizations and code handling features. It has been the author's experience that compilers are often a source of problems, particularly when the code has been compiled for the first time on a specific architecture, or when a new release of the operating system or FORTRAN compiler has been installed. Before reporting an unsolvable problem, it is a good practice to completely recompile the code on a known stable machine with a well tested version of FORTRAN *without* using optimization (the user may be required to modify the *ADPAC08* Makefile to do this). If the code displays the same error, then it is possible that a bug has been uncovered and this should be reported so future versions do not encounter the same problem.

Step 6.) Determine the effect of key input variables

Some “fine-tuning” of input variables is occasionally required to obtain a converged solution, or to prevent an instability from forming. The following suggestions may be useful to aid in establishing the sensitivity of the solution to various inputs:

6.1 Try to run the problem without any boundary conditions. This is essentially a uniform flow test. If the code diverges, then the problem is either in the input file or the mesh.

6.2 Vary the parameter **CFMAX**. Lower values imply more smoothing. It is possible to have too much smoothing, so both larger and smaller values should be tested.

6.3 Make sure **FRESID** is set to 1.0 if the magnitude of **CFL** is larger than 2.0.

6.4 Examine and vary the values of **VIS2** and **VIS4**.

6.5 Turn off all multigrid (**FMULTI, FFULMG** = 0.0).

6.6 Turn off the turbulence model **FTURBB** = 999999999.0. If the problem still exists, try to run inviscid flow (**FINVVI** = 0.0).

6.7 Clear the input file and boundary data file of all specifications (except the case name, which must be activated). Now, if the code diverges, there is almost certainly a problem with the grid. Examine the code output to determine where the maximum error occurs, and carefully check the grid in this region.

Step 7.) Report the Problem

In the event that no other cause of the problem can be detected, the problem should be reported to NASA. The recommended contact is:

```
Dr. Chris Miller
Mail Stop 77-6
NASA-Lewis Research Center
21000 Brookpark Road
```

```
Cleveland, OH 44135
(216) 433-6179
cmiller@lerc.nasa.gov
```

The author is also interested in keeping up with known problems and may be reached at:

```
Dr. Ed Hall
Speed Code T-14A
Allison Engine Company
Indianapolis, IN 46206-0420
(317) 230-3722
ehall@nas.nasa.gov
ieeh1@agt.gmeds.com
```

Chapter 4

RUNNING *ADPAC08* IN PARALLEL

4.1 Parallel Solution Sequence

In order to run *ADPAC08* in parallel, *ADPAC08* must be compiled for parallel execution. The chapter on code compilation in this User's Manual describes the proper compilation procedure.

ADPAC08 is parallelized using the Application Portable Parallel Library (APPL) message passing library, developed at NASA Lewis. Reference [18] explains how to write code using APPL and how to run codes written with APPL. While APPL runs on many platforms, this manual will deal with only two of them: workstation clusters and nCUBE massively parallel computers. These two platforms are representative of how *ADPAC08* runs in parallel. The APPL document should be consulted for cases not covered in this manual.

Regardless of the platform, running *ADPAC08* in parallel requires the APPL **compute** function and a *procdef* file. Codes running under APPL are not initiated by typing the executable name, but use the APPL **compute** function instead. The syntax for executing *ADPAC08* is as follows:

compute < *casename.input* > *output*

The **compute** function controls the execution of *ADPAC08* on the various processors, taking additional input from the *procdef* file. The *procdef* file contains the names of the executable images and the processors that they are to be loaded on. The **compute** function establishes communications with each processor specified in the *procdef* file, loads the *ADPAC08* executable image, and initiates the run on each processor. Also, the **compute** function oversees the running processes, monitoring the processors for abnormal terminations. If a communications error is trapped, or if a process has died unexpectedly, the **compute** function shuts down all of the remaining processes gracefully. This feature is most important on workstation clusters, which have no built-in mechanism for monitoring parallel jobs.

The normal *ADPAC08* input file is redirected from standard input to the **compute** function. The redirected input is available to all of the processes (although *ADPAC08* currently does all reading from node 0). The output file may be redirected, or allowed to stream to the terminal, just as in serial.

The *procdef* file should appear in the directory where the job is being run. It has a different syntax for the various parallel platforms. The simplest formulation is for hypercube machines (nCUBE and Intel). A sample *procdef* file for an nCUBE 2 is as follows:

```
someuser frntend . ./adpacp.ncube -1 32
```

The first token in the procdef file is the user name (someuser). The second token is the name of the front-end processor to the nCUBE 2. The third token is the path to the directory for input and output files (in this case, the current directory, "." is used). The fourth token is the executable name (the path may be specified to be sure the correct executable is used). The fifth token specifies how the processors are mapped (-1 indicates hypercube ordering, -2 indicates mapping into a ring). Hypercube ordering is generally preferred. The last token specifies the number of processors to be allocated (32 in this case).

Similarly, a sample *procdef* file for a workstation cluster is as follows:

```
someuser  host1   . 1 adpacp.aix
someuser  host2   . 1 adpacp.aix
someuser  host3   . 2 adpacp.aix adpacp.aix
```

In this example, the first three tokens represent the user name, host name and the path to the working directory, just as before. The fourth token indicates the number of processes to be run on the host, and the remaining tokens are the executable images corresponding to the processes. The last line of the example shows 2 processes running on host3. Using this *procdef* file, the virtual parallel computer will consist of four processes running on three workstations.

The host machines in a workstation cluster must be connected by ethernet, but do not have to share disks, or be part of the same subnet. This provides tremendous flexibility in constructing a workstation cluster. However, most performance bottlenecks encountered on workstation clusters involve the network. The benefits of adding processors may be offset by poor network performance. The tradeoff varies with the problem and with the hardware configuration.

In general, the behavior of *ADPAC08* in parallel is the same as in serial. This is especially true if there are no input errors. The output files may be different if there are input errors. There are two general types of input errors detected in *ADPAC08* . Errors involving the grid or the input file will generally be detected by all processors, and the error messages will appear as they do in serial.

If, however an error is discovered in a boundary condition routine, the output messages will probably appear differently in the output file, and may not appear at all. Since *ADPAC08* boundary conditions are applied in parallel, node 0 does not execute all of them, but only those involving a block assigned to node 0. If node 0 does not encounter the error, then a different node writes the error message. Since the writing node is out of sync with node 0, the error message may be written to a different place in the output file than if node 0 had written it.

Buffering of output on the various processors can also cause a problem. Usually, after an error message is printed, execution is stopped on all processors. If execution is stopped before the buffer is flushed, then output may be lost from some processors. The result is that an error message could be caught in the buffer and never appear in the output file. If *ADPAC08* terminates for no apparent reason, this may explain the problem. The solution is to rerun the job without redirecting the output. If output is not redirected, it is normally not buffered, and all of the output will appear.

It is also possible to get multiple copies of an error message if more than one processor encounters the error. Wherever possible, *ADPAC08* has been coded to avoid these problems, but these unfortunate possibilities still exist. Therefore, running *ADPAC08* interactively is the best way to track down input problems.

Aside from these considerations, running *ADPAC08* in parallel is very much like running *ADPAC08* in serial. The input files are identical, and the output files are very similar. The most common problems in running *ADPAC08* in parallel are failing to use the **compute** function, improperly specifying the parallel configuration in the procdef file, and attempting to run a serial executable in parallel.

4.2 *SIXPAC* (Block Subdivision) Program

SIXPAC , which stands for **S**ubdivision and **I**nformation e**X**change for **P**arallel **A**dpac **C**alculations, enables the user to redefine the block structure of an *ADPAC08* job. Using *SIXPAC* , large grid blocks can be subdivided to improve load balance, or to make use of smaller memory processors in parallel calculations. *SIXPAC* generates new input, mesh, restart, and boundata files for the subdivided problem, creating new blocks according to user specifications. The resulting files represent a problem equivalent to the original, but with more, smaller, blocks. Although the number of unique grid points is unchanged, the total number of points is larger because of duplication at interfaces.

The motivation for *SIXPAC* comes from the way *ADPAC08* was parallelized. Rather than parallelize the interior point solver, *ADPAC08* was parallelized through the boundary conditions. An individual block can't be run across multiple processors; each processor must contain only whole blocks. This implies that a problem with a single large block couldn't be run in parallel. *SIXPAC* enables large blocks to be recast as groups of smaller blocks, so that they can be run in parallel. *SIXPAC* is not required to run a problem in parallel, but it simplifies the process of setting up a problem for optimal parallel performance.

4.2.1 *SIXPAC* Input

The input files required by *SIXPAC* are the *casename.input* file, the *casename.mesh* file, the *casename.boundata* file, and the *casename.sixpac* file. If a new restart file is to be created, then a *casename.restart.old* file is also required. Of this group, only the *casename.sixpac* file is different from the standard *ADPAC08* input.

The *casename.sixpac* file contains information which specifies how the blocks are to be subdivided. The required information includes the number of original blocks, and how each block is to be subdivided in each indicial direction (i, j, and k). In each direction, the number of subdivided blocks, and possibly the locations of the subdivisions, must be specified. If the number of subdivided blocks in a particular coordinate direction is set to 1, then the block is not divided in that coordinate direction.

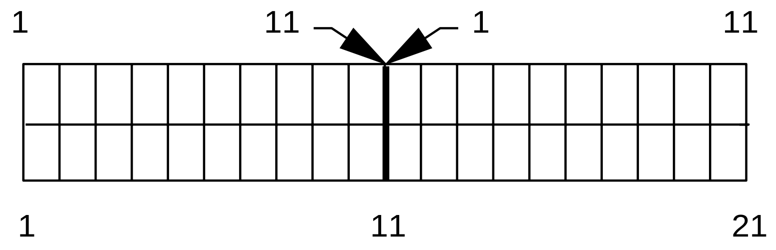
By default, blocks are split into the specified number of equal sized pieces. If there is a remainder, it is spread over the processors to create nearly equal sized pieces. If unequal divisions are required in a particular direction, then the location of each division must be specified in that direction.

Unequal divisions are often employed to preserve levels of multigrid, or to put the edge of a geometric feature on a block boundary. Figure 4.1 illustrates how different block strategies affect multigrid.If, for example, there are 21 points in the I direction of a block, 3 levels of multigrid are possible. If this block is divided into two equal pieces of 11 points each, then only 2 levels of multigrid are possible. However, if the block is split into a block with 13 points and a block with 9 points, 3 levels of multigrid are still possible.

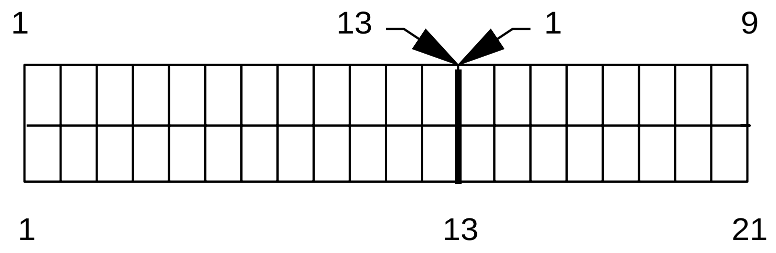
4.2.2 *casename.sixpac* File Contents

For equal divisions of the blocks in each direction, the *casename.sixpac* is simple to construct. The first line is a comment, and the second line contains the number of blocks in the original problem. Input is free format. The third line is a comment, and there is an additional line for each original block, in ascending order. These lines contain the block number, and the number of subdivided blocks in each coordinate direction. The following is a sample *casename.sixpac* file:

Subdivision of Blocks to Preserve Levels of Multigrid



Subdivision into two equal pieces results in blocks with 11 points. Only two levels of multigrid are possible, even though three levels were possible for the original block.



Subdivision into two unequal blocks, one with 13 points and one with 9 points, yields a grid capable of three levels of multigrid, like the original block.

Figure 4.1: Careful block division can preserve levels of multigrid.

```
Number of blocks
  2
 n   idiv  jdiv  kdiv
 1      4     2     1
 2      4     2     1
```

In this example, there are two original blocks. The first block is to be divided into 4 pieces along the I coordinatem, 2 pieces along the J coordinate, and 1 piece along the K coordinate. The second block is to be divided into 4 pieces along the I coordinatem, 2 pieces along the J coordinate, and 1 piece along the K coordinate. This means that there will be a total of 16 new blocks generated from the original 2 blocks.

If, however, user-specified divisions are required in a direction, the *casename.sixpac* file must be modified as follows:

- The number of subdivided blocks in the direction to be specified is set to 0. This tells *SIXPAC* that user specifications are to follow.
- New lines are added to the *casename.sixpac* immediately following the block to be modified. First, a comment line is added, which normally identifies which direction is being specified. Second, a line containing the number of subdivided blocks is specified (either nblki, nblkj, or nblkk, depending on the direction) Third, a comment line is added, which normally indicates that the following line contains block division points. Fourth, lines are added containing the division positions for the new blocks. There should be nblki, nblkj, or nblkk of these entries, as specified above, in free-format.
- The block division positions are the upper limits of the new blocks in terms of the original block indices. The last division should be the block size in that direction.
- Block division positions must be specified in ascending order.
- If user-specified subdivisions are required in more than one direction of a single block, then the additions are made in “natural order,” that is I first, then J and K, as required.

- All blocks must appear in the *casename.sixpac* file in ascending order.

SIXPAC has some sanity checks built in to warn users of problems in the *casename.sixpac* file. A sample *casename.sixpac* file appears below.

```
Number of blocks
2
n   idiv jdiv kdiv
1      4    0    1
   number of J divides
   2
   J break points
    3 10
2      4    2    1
```

As before, there are two original blocks. The first is to have 4 I divisions, 2 J divisions and 1 K division. The J divisions are to be at J=3 and J=10 in block 1. The second block is to have 4 I divisions 2 J division and 1 K division. This means that there will be a total of 16 new blocks generated from the original 2 blocks.

4.2.3 Restart Files in *SIXPAC*

If a restart file is to be created for the subdivided problem, the input trigger **FREST** must be set equal to 1.00 in the *casename.input* file. This tells *SIXPAC* to look for a *casename.restart.old* file, and to subdivide it. A *Ncasename.restart.old* file is written, and the new input file will be set up to run with the new restart file.

4.2.4 *SIXPAC* Output

The output files produced by *SIXPAC* are the *Ncasename.input* file, the *Ncasename.mesh* file, the *Ncasename.boundata* file, and the *Ncasename.bacpac* file. An *Ncasename.restart.old* file is also created if required. The casename has been prepended by an "N" to avoid confusion with the original input files. The new output files are themselves *ADPAC08* input files and can be run in either serial or parallel versions of *ADPAC08* .

The *Ncasename.bacpac* file is not required to run *ADPAC08* , but is used by the code *BACPAC* , which reassembles the blocks into their original, undivided structure. The *Ncasename.bacpac* file contains information about the way *SIXPAC* subdivided the blocks. There is normally no reason for the user to alter the *Ncasename.bacpac* file. The form of the *Ncasename.bacpac* is described in the section of the User's Manual dealing with *BACPAC* input.

4.2.5 Running *SIXPAC*

Running *SIXPAC* is very much like running *ADPAC08* . The command syntax is:

sixpac < *casename.input* > *output*

The *output* file is similar to an *ADPAC08* output file, because the routines to read the grid, the input file and the boundary data file are taken directly from *ADPAC08* . One addition to the output file is a table of the new grid blocks and their sizes. After verifying the new block structure created by *SIXPAC* , the output file can be discarded.

4.3 *BACPAC*

BACPAC , which stands for **B**lock **A**ccumulation and **C**onsolidation for **P**arallel **A**dpac **C**alculations, reassembles subdivided *ADPAC08* files into their original, undivided form. It is used in conjunction with *SIXPAC* , and performs essentially the inverse operation of *SIXPAC* . *BACPAC* can reconstruct mesh, PLOT3D, or restart files, producing new files which are equivalent to what would have been produced had the problem been run with the original, undivided blocks. Using *SIXPAC* and *BACPAC* , a problem can be subdivided and reconstructed any number of ways to take advantage of available computer resources.

4.3.1 *BACPAC* Input

BACPAC queries the user for needed information, and reads from standard input (normally the keyboard). The user is first prompted for the casename. The user then selects which files are to be reconstructed by entering appropriate responses to questions about each file. Due to the potential size of these files, they are not created by default.

BACPAC expects to find a *casename.bacpac* file which contains information detailing how the original problem was subdivided. The *casename.bacpac* file is created automatically by *SIXPAC* , and requires no modifications by the user. However, if *SIXPAC* was not used to create the subdivided blocks, the user must construct a *casename.bacpac* file in order to run *BACPAC* . A sample *casename.bacpac* file resulting from the first sample *SIXPAC* input file given previously appears below.

```
          2   original number of blocks
   imax      jmax      kmax
     73        10         9
  nblki     nblkj     nblkk
      4         2         1
 oldblk    newblk  global i  global j  global k  local im  local jm  local 
     73        10         9
  nblki     nblkj     nblkk
      4         2         1
 oldblk    newblk  global i  global j  global k  local im  local jm  local 
```

In the above example, two blocks are subdivided into eight new blocks each (a total of 16 blocks). The dimensions of the original blocks are $73x10x9$, and there are 4, 2, and 1 subdivided blocks in each coordinate direction for each block. The table underneath each of the original block size declarations shows the original block number, and the new block number. The global i, j, and k indices are the position of the bottom right hand corner of the new block in the original block. For example, the point (1,1,1) in the new block 8 is the same as the point (17,9,9) in the original block 1. The local im, jm, and km indices are the block size of the new block. This data essentially maps the new blocks into the original block structure.

4.3.2 *BACPAC* Output

The output files produced by *BACPAC* are the *Ncasename.mesh.bac* file, the *Ncasename.p3dabs.bac* and the *Ncasename.p3drel.bac* files, and the *Ncasename.restart.bac* file. The *.bac* suffix is used to avoid confusion with existing files. Generally the *Ncasename.mesh.bac* need not be created because it is identical to the original *casename.mesh* file.

4.4 Parallel *ADPAC08* Block/Processor Assignment

Load balancing is a critical issue for parallel computing tasks. While it is beyond the scope of this program to perform detailed load balancing analyses for every parallel computing platform tested, it seems reasonable to provide some form of control in order to distribute computational tasks efficiently across a parallel computing network. In the parallel *ADPAC08* code, this is best accomplished through manipulation of the block/processor distribution scheme. By default, the parallel operation of the *ADPAC08* code provides an automatic block to processor assignment by dividing up the blocks as evenly as possible, and, to the greatest degree possible, assigning sequential block numbers on a given processor. For example, if 8 blocks were divided between 3 processors, blocks 1, 2, and 3 would be assigned to process #0, blocks 4, 5, and 6 to processor #1, and blocks 7, and 8 to processor #2 (note that the processor numbering scheme is 0, 1, 2, etc.). This procedure is nearly optimal when each block is the same size, and each processor has the same computational power. Unfortunately, our experience is that block sizes and computational resources often vary dramatically. In this regard, a system was developed which permits the user to specify the block to processor assignment through a special input file (*casename.blkproc*). A sample *casename.blkproc* file is given below for an 8 block mesh distributed across 3 processors:

```
number of blocks
8
block #      proc #
```

```
1           0
2           1
3           1
4           1
5           1
6           2
7           2
8           2
```

In the case described by the above file, block 1 is assigned to processor #0, blocks 2, 3, 4, and 5 to processor #1, and blocks 6, 7, and 8 to processor #2. This block assignment might be advisable for the case when block 1 is significantly larger in size than the other blocks, or if processor #0 has less memory or a slower CPU than the remaining processors. The original block assignment scheme is selected as the default when the *casename.blkproc* file is not present.

Chapter 5

ADPOST POST PROCESSOR OPERATING INSTRUCTIONS

5.1 Introduction to *ADPOST*

In this chapter, a brief description of the *ADPOST* post-processing routine used to calculate a number of integrated quantities presented in the companion Final Report [1] is given. Early in the development of the *ADPAC08* code, it was determined that the most effective way to handle output data was to utilize a separate, efficient data processing routine, to free the *ADPAC08* code from the expense and complexity of calculating specific case-dependent output data. The wide variety of applications possible with the multiple-block code simply prohibited embedding a number of output data summaries in the aerodynamic analysis, and this prompted the development of the *ADPOST* program. *ADPOST* is designed to handle a number of post processing chores including mass-averaging and calculating radial distributions of flow data. Many of the options in *ADPOST* are slanted towards turbomachinery flows, but the program may be used with equal effectiveness on non-turbomachinery flows.

5.2 Configuring *ADPOST* Maximum Array Dimensions

The maximum array dimensions of the *ADPOST* program are set in exactly the same manner as the *ADPAC08* program itself. All array size PARAMETER statements are in the source file **parameter.inc** included with the *ADPOST* program. A sample **parameter.inc** file would appear as:

```
parameter   ( nbmax  =      6   )
parameter   ( nra3d  = 350000   )
```

Here again, as in *ADPAC08* , NBMAX is the maximum number of mesh blocks permitted (6, in this case), and NRA3D is the maximum sum of all cell centered elements required for the mesh (a formula for estimating this value is given in Section 3.3). An error message is displayed whenever the user attempts to read in a mesh which is larger than that permitted by either NBMAX or NRA3D.

5.3 Compiling the *ADPOST* Program

In the directory containing the FORTRAN source of the *ADPOST* code, compilation is performed by executing the command:

make *option*

The **make** command is standard on UNIX systems and automatically interrogates the file *Makefile* for instructions on how to perform the compilation. The *option* argument may be any of the variables listed below:

No argument - same as *link* below.

link This is the standard UNIX system compilation. This option will deliver a working executable on most UNIX systems which support standard naming conventions (*f77* as the standard compiler, etc.). The compilation includes basic compiler optimization (f77 -O).

cray This option is utilized when compiling the *ADPOST* code on a Cray computer.

aix This option is used when compiling the *ADPOST* code on an IBM RS-6000 workstation running the AIX operating system.

5.4 Running the *ADPOST* Program

Once the code has been compiled, change directories to the location where the case of interest is stored. The *ADPOST* program is invoked by issuing the command:

*path/*__adpost__

where *path* is the relative or absolute pathname of the directory containing the *ADPOST* executable file from the current local directory. For example, if the mesh and flow files are in the directory

```
/usr/people/me/testcase
```

and the *ADPOST* executable is in the directory

```
/usr/people/me/adpac/src/adpost
```

then the commands

```
cd /usr/people/me/testcase
/usr/people/me/adpac/src/adpost/adpost
```

would begin the *ADPOST* program process.

Once the *ADPOST* program is invoked, the following message appears:

```
*************************************************

    aaaa     dddd   ppppp   ooooo   sssss   ttttt
   a   a   d   d   p   p   o   o   s         t
  aaaaa   d   d   ppppp   o   o   ssssss    t
 a   a   d   d   p       o   o        s    t
a   a   dddd    p       ooooo   ssssss    t



*************************************************




Welcome to ADPOST, a post-processor for the
ADPAC code.  All that is required to run this
routine is the original input file and the
corresponding mesh, and restart.old files.

You will first be prompted for the input
filename.  It must contain the casename that
precedes the other files.

You will then be presented with several post
processing options from which you may choose.



Enter the Name of the Input File to be Post Processed.
 (be sure to enter both the filename and its suffix)
```

The input filename must now be entered and must contain the *case* name to be post-processed. The original input file, used with the *ADPAC08* code is recommended since it is in the expected format and contains the information necessary for dimensionalizing flow variables. Next the restart.old and mesh data are read, after which the following menu appears:

```
***************************************************

Choose one of the following options to continue:


            MAIN OPTION MENU
           ------------------

  1. Specify a section to be averaged.

  2. Dimensionalize flow values.

  3. Print relative quantities instead of absolute.

  4. Print spanwise information also.

  5. Maintain thermodynamic consistency.

  6. Plot flow variables.

  7. Exiting  Adpost.


***************************************************

Enter option number now.
```

Currently, only six options are available since the plot routines for option #6 have not yet been incorporated. A brief description of each of the other six options follows:

```
1. Specify a section to be averaged.
```

Option #1 allows the user to specify the portion of the flow field for which the mass averaged quantities are to be calculated. This value can only be specified for constant i, j or k planes but need not include the entire flow field on these planes. If no other options are picked in conjunction with #1 then, by default, the mass averaged values calculated for the specified section are in nondimensional form; the velocities and stagnation properties are absolute values, and thermodynamic consistency is maintained in each grid cell but not necessarily in the resulting mass averaged quantities. Other formations are possible by choosing additional options in combination with #1. Only one section of the flow field can be mass averaged at a time, so if values are needed in separate sections/blocks this option must be chosen repeatedly. Mass averaged quantities calculated with *ADPOST* are; stagnation temperature and pressure, static temperature and pressure, and velocity. A sample output is shown below:

```
 FLOW QUANTITIES FOR BLOCK #1
 ----------------------------


 MASS AVERAGING OVER i = CONSTANT SLICES


 GRID              MASSFLOW     TOTAL       TOTAL       STATIC     STATIC
 SLICE     AREA       RATE     PRESSURE  TEMPERATURE  PRESSURE TEMPERATURE

   1     0.1282E-01 0.2135E-02 0.1033E+01 0.1000E+01 0.9949E+00 0.9894E+00
   2     0.1282E-01 0.2135E-02 0.1033E+01 0.1000E+01 0.9949E+00 0.9894E+00
   3     0.1282E-01 0.2135E-02 0.1033E+01 0.1000E+01 0.9949E+00 0.9894E+00
   4    0.1282E-01 0.2135E-02 0.1033E+01 0.1000E+01 0.9949E+00 0.9894E+00
        etc.
```

If no section if specified with option #1, then calculations for all other options, except #7, are performed on the entire flow field.

```
2. Dimensionalize flow values.
```

Option #2 dimensionalizes mass averaged flow values with the reference pressure, temperature, and diameter found in the input file. For a description of how variables are dimensionalized, see Section 1.2 of the Final Report [1].

```
3. Print relative quantities instead of absolute.
```

Option #3 calculates the mass averaged velocity, total pressure and total temperature relative to a rotating coordinate system (i.e., rotor).

```
4. Print spanwise information also.
```

Option #4 provides additional mass averaged values for each radial (spanwise, j index) location in the specified planes. Unless radial planes are specified, then values at each axial location are given instead. The spanwise/axial averages are printed first, followed by the values for the entire plane. A sample output is shown below:

```
 FLOW QUANTITIES FOR BLOCK #1
 ----------------------------


 MASS AVERAGING OVER i = CONSTANT SLICES
 spanwise integration for grid surface i =   1

j slice # percent span    mass flow    ptotal       ttotal     swirl angle
     2   0.000 - 0.001  0.265201E-06  1.00322      1.00140     -71.9231
     3   0.001 - 0.001  0.733345E-06  1.00299      1.00024     -60.4713
     4   0.001 - 0.003  0.112133E-05  1.00344     0.999918     -57.8162
     .
     .
     .
     .

 Mass Average Totals for i=  1 Plane

 GRID               MASSFLOW     TOTAL      TOTAL       STATIC    STATIC
 SLICE     AREA       RATE     PRESSURE  TEMPERATURE  PRESSURE TEMPERATURE

   1    0.1282E-01 0.2135E-02 0.1033E+01 0.1000E+01 0.9949E+00 0.9894E+00
```

`5. Maintain thermodynamic consistency.`

Option #5 maintains thermodynamic consistency in the averaged flow quantities. This is accomplished by mass averaging only the cell centered conservation variables. The resulting mass averaged velocity, density and energy are then used with perfect gas relationships to calculate pressure and temperature.

`6. Plot flow variables.`

This option is not currently available.

`7. Exiting Adpost.`

After each option has been chosen, the user is asked if they would like to specify any additional options. If the response is "yes", then you are returned to the main menu. If the response is “no” then calculations are performed on the specified section with the options chosen, after which you are again returned to the main menu. The user may then either specify another section or other options to be performed.

The entire main menu sequence is repeated until option #7 is picked and the process is terminated. Calculated quantities are stored in a file called *adpost.out*.

5.5 Sample Session Using the *ADPOST* Program

A sample interactive session from the *ADPOST* program is printed below for the case of a ducted propfan H-grid with multiple blade rows (Standard Configuration #7). The mesh configuration and mesh index reference are described in the Appendix for the sample test case. All the user responses are listed in boldface type during the interactive session listing.

Sample Interactive *ADPOST* Session Listing

```
*************************************************

    aaaa     dddd   ppppp   ooooo   sssss   ttttt
   a   a   d   d   p   p   o   o   s         t
  aaaaa   d   d   ppppp   o   o   ssssss    t
 a   a   d   d   p       o   o        s    t
a   a   dddd    p       ooooo   ssssss    t
```

```
*************************************************




Welcome to ADPOST, a post-processor for the
ADPAC code.  All that is required to run this
routine is the original input file and the
corresponding mesh, restart.old, p3dabs, and
p3drel files.

You will first be prompted for the input
filename.  It must contain the casename that
precedes the other files.

Next, mass-average flow values are calculated
at each axial station.

Finally, the user is presented with several
other post processing options from which they
may choose.



Enter the Name of the Input File to be Post Processed.
 (be sure to enter both the filename and its suffix)
```

nasa.input

```
Mass averaged values are now being
calculated for every 'i' plane.



*****************************************************
```

```
Calculations have been written to 'adpost.out'.
-----------------------------------------------



Choose one of the following options to continue:

           MAIN OPTION MENU
          -----------------

  1. Specify another section to be averaged.

  2. Dimensionalize flow values.

  3. Print relative quantities instead of absolute.

  4. Print spanwise information also.

  5. Maintain thermodynamic consistency.

  6. Plot flow variables.

  7. Exit Adpost.

Enter option number now.
```

1

```
**************************************************

  Option #1 allows you to specify the portion of
  flow field to be averaged.  This can be done on
  any i, j or k plane and need not include the
```

```
   the entire slice.



Enter the block number to be mass-averaged
or if you wish to return to the Main Menu,
enter a number greater than  4
```

1

```
In what direction are slices to be averaged?
 (Enter i, j, or k)
```

i

```
Specify the section to be averaged


Enter the minimum i,j,k, indices  of the first surface.
```

129 1 1

```
Enter the maximum i,j,k, indices of the last surface.
Maximum grid indices:  129, 17, 17
```

129 17 17

```
Enter the increment in the i direction.
```

1

```
 Would you like to specify any additional
 options?  (yes/no)
```

yes

```
 Choose one of the following options to continue:

             MAIN OPTION MENU
            ------------------

   1. Specify another section to be averaged.

   2. Dimensionalize flow values.

   3. Print relative quantities instead of absolute.

   4. Print spanwise information also.

   5. Maintain thermodynamic consistency.

   6. Plot flow variables.

   7. Exit Adpost.

 Enter option number now.
```

5

```
 *******************************************

 Thermodynamic consistency will be preserved.

 *******************************************


 Would you like to specify any additional
 options?  (yes/no)
```

yes

```
 Choose one of the following options to continue:

            MAIN OPTION MENU
           ------------------

   1. Specify another section to be averaged.

   2. Dimensionalize flow values.

   3. Print relative quantities instead of absolute.

   4. Print spanwise information also.

   5. Maintain thermodynamic consistency.

   6. Plot flow variables.

   7. Exit Adpost.

 Enter option number now.
```

4

```
 *******************************************

 Spanwise print flag has been turned on.

 *******************************************


 Would you like to specify any additional
 options?  (yes/no)
```

no

```
 Calculations for selected options are now being processed.
 ----------------------------------------------------------


 Calculations are complete and can be found in 'adpost.out'.
 -----------------------------------------------------------


 Choose one of the following options to continue:

             MAIN OPTION MENU
            ------------------

   1. Specify another section to be averaged.
   2. Dimensionalize flow values.

   3. Print relative quantities instead of absolute.

   4. Print spanwise information also.

   5. Maintain thermodynamic consistency.

   6. Plot flow variables.

   7. Exit Adpost.

 Enter option number now.
```

7

5.6 Sample Output File from the *ADPOST* Program

The output file **adpost.out** from the interactive session given in Section 4.5 is listed below. Note that the output is similar to the *ADPAC08* program initially, as *ADPOST* reads in the input, mesh and restart files. At the end of the file, the various summations from the mass-averages requested in the interactive session are displayed.

```
 ************************************************
```

```
    aaaa     dddd   ppppp   ooooo   sssss   ttttt
   a   a   d   d   p   p   o   o   s         t
  aaaaa   d   d   ppppp   o   o   ssssss    t
 a   a   d   d   p       o   o        s    t
a   a   dddd    p       ooooo   ssssss    t



*************************************************



 case name defined as: nasa
 *********************


**************************************************
               begin grid input
**************************************************


 mesh file name: nasa.mesh
 ***************

 number of grids: (nblks) =            4
 ****************

 grid block dimensions :
 ***********************

     block       imx       jmx       kmx
     -----       ---       ---       ---
         1       129        17        17
         2       129        17        17
         3        97        17        17
         4        97        17        17
 initialize storage array and pointer locations
 **********************************************

      block     3-d ijk     length    total length
        1           1       42120       42120
        2       42121       42120       84240
        3       84241       31752      115992
        4      115993       31752      147744

 program array dimensions:
 ************************

 nra3d (3-d array size) =       350000


 begin reading grid block coordinates
 ***********************************

grid #            1 size          130          18           18
 grid block           1 coordinates successfully read
 conatan routine used for cylindrical conversion
grid #            2 size          130          18           18
 grid block           2 coordinates successfully read
 conatan routine used for cylindrical conversion
grid #            3 size           98          18           18
 grid block           3 coordinates successfully read
 conatan routine used for cylindrical conversion
grid #            4 size           98          18           18
 grid block           4 coordinates successfully read
 conatan routine used for cylindrical conversion

*************************************************
              begin flow input
*************************************************
```

```
    input file read on unit  15
    **********************


    input file data
    ***************
     rm     ( reference mach number          ) =     0.7500
     gamma  ( specific heat ratio            ) =     1.4000
     cfl    ( cfl number (+ time accurate    ) =     5.0000
     vt     ( =1, time acc., =0, local time  ) =     0.0000
     vis2   ( 2nd order dissipation factor   ) =     0.5000
     vis4   ( 4th order dissipation factor   ) =     0.0156
     fncmax ( maximum iterations             ) =    10.0000
     rest   ( 1 = restart solution           ) =     0.0000
     save   ( 1 = save solution              ) =     0.0000
     alpha  ( angle of attack                ) =     0.0000
     fistep ( recalculate time step          ) =     1.0000
     fnprnt ( 1 = print flowfield            ) =     1.0000
     fiprnt ( 1 = print partial flowfield    ) =     1.0000
     finvvi ( 0 = inviscid flow solution     ) =     0.0000
            ( 1 =  viscous flow solution     ) =
     epsx   ( i implicit smoothing factor    ) =     1.5000
     epsy   ( j implicit smoothing factor    ) =     1.5000
     epsz   ( k implicit smoothing factor    ) =     1.5000
     iunint ( # it. for unsteady plot3d file ) =  9999.0000
     itimei ( #it between time step update   ) =     1.0000
     iturbi ( #it between turb. model update ) =     1.0000
     iturbb ( #it before starting turb model ) =  9999.0000
     diam   ( blade diameter (ft)            ) =     9.0000
     tref   ( total temperature (deg. r)     ) =   518.7000
     pref   ( total pressure    (lbf/ft**2)  ) =  2116.8000
     rgas   ( gas constant (ft-lbf/lbm-deg r)) =  1716.2600
     pr     ( prandtl number                 ) =     0.7000
     prt    ( turbulent prandtl number       ) =     0.9000


non-dimensional reference values calculated as:
```

```
***********************************************
     rho0   ( reference density             ) =     0.7660
     u0     ( reference axial velocity      ) =     0.8413
     v0     ( reference radial velocity     ) =     0.0000
     w0     ( reference theta velocity      ) =     0.0000
     ei0    ( reference internal energy     ) =     2.6011
     h0     ( reference enthalpy            ) =     3.5000
     p0     ( reference pressure            ) =     0.6886
     t0     ( reference temperature         ) =     0.8989
     dmu0   ( reference viscosity           ) =     0.9195
     omega  ( rotational speed              ) =    -1.9016


non-dimensional value for row   2 calculated as:
***************************************************
     omega  ( rotational speed              ) =    -1.9016


non-dimensional value for row   3 calculated as:
***************************************************
     omega  ( rotational speed              ) =     0.0000


non-dimensional value for row   4 calculated as:
***************************************************
     omega  ( rotational speed              ) =     0.0000


dimensional reference values calculated as:
*******************************************
     rho0   ( density          (lbm/ft**3)  ) = 0.182151E-02
     u0     ( axial velocity   (ft/s)       ) =  793.825
     v0     ( radial velocity  (ft/s)       ) = 0.000000E+00
     w0     ( theta velocity   (ft/s)       ) = 0.000000E+00
     ei0    ( total int. energy (ft-lbf/lbm) ) = 0.231558E+07
     h0     ( total enthalpy   (ft-lbf/lbm) ) = 0.311578E+07
     p0     ( total pressure   (lbf/ft**2)  ) =  1457.57
     t0     ( total temperature (deg. r)     ) =  466.247
     dmu0   ( viscosity         (lbf-s/ft**2)) = 0.343739E-06
     omega  ( rotational speed  (rad/s)      ) = -199.351
     rpm    ( rev/min           (rev/min)    ) = -1903.66


additional blade row   2 values calculated as:
***********************************************
     omega  ( rotational speed  (rad/s)      ) = -199.351
     rpm    ( rev/min           (rev/min)    ) = -1903.66


additional blade row   3 values calculated as:
***********************************************
     omega  ( rotational speed  (rad/s)      ) = 0.000000E+00
     rpm    ( rev/min           (rev/min)    ) = 0.000000E+00


additional blade row   4 values calculated as:
***********************************************
     omega  ( rotational speed  (rad/s)      ) = 0.000000E+00
     rpm    ( rev/min           (rev/min)    ) = 0.000000E+00


nondimensionalization values calculated as:
*******************************************
     rho    ( density           (lbm/ft**3)  ) = 0.237783E-02
     u      ( axial velocity    (ft/s)       ) =  943.517
     ei     ( total int. energy (ft-lbf/lbm) ) =  890224.
     p      ( total pressure    (lbf/ft**2)  ) =  2116.80
     t      ( total temperature (deg. r)     ) =  518.700
     dmu    ( viscosity         (lbf-s/ft**2)) = 0.373852E-06
     omega  ( rotational speed  (rad/s)      ) =  104.835
     length ( length            (ft)         ) =  9.00000


  input data successfully entered
```

```
 ******************************


 set up block data:
 *****************

 cell area terms calculated for grid block   #           1
 cell area terms calculated for grid block   #           2
 cell area terms calculated for grid block   #           3
 cell area terms calculated for grid block   #           4

 cell volume terms calculated for grid block #           1
 cell volume terms calculated for grid block #           2
 cell volume terms calculated for grid block #           3
 cell volume terms calculated for grid block #           4

**************************************************
          begin restart input
**************************************************


 restart file name: nasa.restart.old
 *****************

 number of blocks (nblksf):           4
 *************************


 restart file block sizes
 ***********************

    block         imx          jmx          kmx
        1         130          18          18
        2         130          18          18
        3          98          18          18
        4          98          18          18
```

```
  restart data for block #            1 entered
  restart data for block #            2 entered
  restart data for block #            3 entered
  restart data for block #            4 entered


 FLOW QUANTITIES FOR BLOCK #1
 ----------------------------


 MASS AVERAGING OVER i = CONSTANT SLICES


 GRID              MASSFLOW     TOTAL      TOTAL      STATIC    STATIC
 SLICE     AREA      RATE     PRESSURE  TEMPERATURE  PRESSURE TEMPERATURE



 spanwise integration for grid surface i = 129

j slice # percent span    mass flow    ptotal       ttotal     swirl angle
     2   0.000 - 0.022  0.388604E-03  1.12050      1.04476     -27.5691
     3   0.022 - 0.052  0.559818E-03  1.12535      1.04529     -27.8464
     4   0.052 - 0.094  0.799983E-03  1.12826      1.04510     -27.8023
     5   0.094 - 0.148  0.113206E-02  1.13259      1.04478     -27.1676
     6   0.148 - 0.217  0.156474E-02  1.13696      1.04483     -26.2078
     7   0.217 - 0.301  0.208925E-02  1.14238      1.04527     -25.0896
     8   0.301 - 0.397  0.264947E-02  1.14800      1.04656     -24.0559
     9   0.397 - 0.500  0.315033E-02  1.15495      1.04885     -23.3043
    10   0.500 - 0.603  0.347579E-02  1.16192      1.05161     -22.6446
    11   0.603 - 0.699  0.352727E-02  1.16726      1.05458     -22.2051
    12   0.699 - 0.783  0.329510E-02  1.16925      1.05701     -21.8015
    13   0.783 - 0.852  0.285540E-02  1.16869      1.05906     -21.4849
    14   0.852 - 0.906  0.231871E-02  1.16527      1.06065     -21.3212
    15   0.906 - 0.948  0.179275E-02  1.16143      1.06218     -21.1105
    16   0.948 - 0.978  0.133604E-02  1.15755      1.06373     -20.9239
    17   0.978 - 1.000  0.972663E-03  1.15467      1.06496     -20.6821
 129    0.5499E-01 0.3191E-01 0.1156E+01 0.1053E+01 0.9246E+00 0.9882E+00
  processing completed
  execution normally terminated
```

Chapter 6

STANDARD MESH BLOCK CONFIGURATIONS

In this section, a description of several standard mesh block configurations for ducted/unducted fans and other turbomachinery-related geometries are given. The standard configuration list defines a series of pretested mesh block configurations for various turbomachinery-related calculations of interest to gas turbine engine designers. The standard configurations, in conjunction with the *SETUP* and *ROTGRID* programs described in Chapters 6 and 8, respectively, attempt to remove some of the burdens of data preparation involved in an *ADPAC08* solution for inexperienced users. This particular configuration list does not imply any limitations on the *ADPAC08* code itself. In fact, the user is encouraged to try alternate mesh configurations within the limitations of the *ADPAC08* multiple block boundary condition application strategy.

6.1 Description of Standard Configurations

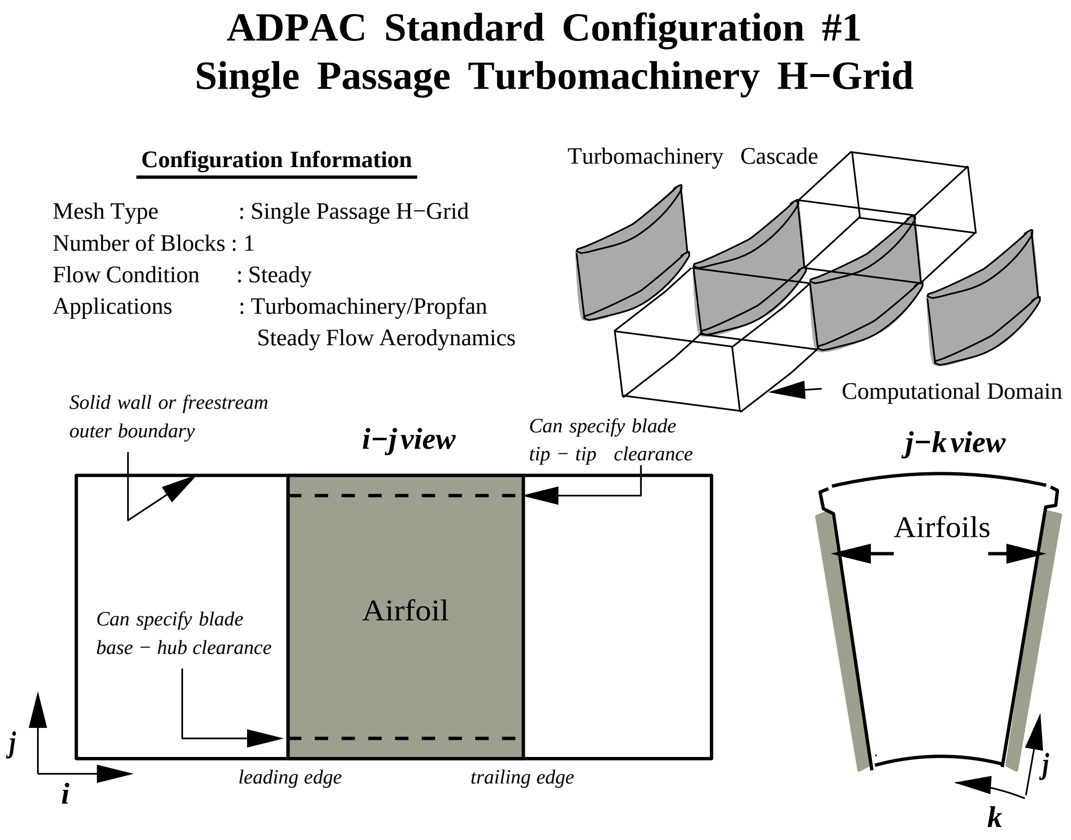
Each of the predefined standard configurations are described below. A corresponding graphic describing the individual configurations are given in Figures 6.1-6.10. In the description of a given configuration, the mesh size is assumed to be represented by the values $imax, jmax, kmax$ as limits in the i, j, and

k coordinate directions, respectively.

Standard Configuration #1

Title:	Single-passage turbomachinery H-grid
Mesh Type:	Single Block H-grid
Number of Mesh Blocks:	1
Flow Type:	Steady
Geometry:	Compressor rotor, stator, unducted fan, propfan
Number of Blade Rows:	1
Mesh Generation Program:	*TIGG3D, CHGRIDV2, MULAC*, and others

Standard Configuration #1 consists of a single H-type mesh block discretizing a single blade passage of a turbomachinery blade row or propfan, as shown in Figure 6.1. The mesh is spatially periodic, except in the vicinity of the blades themselves. The blades are represented as a circumferential displacement on the periodic surfaces (k=1 and k=kmax). Blade hub/tip clearances may be represented by removing the blade displacement in the clearance region above or below the blade. Inner boundaries on this mesh include the blade leading and trailing edge i indices, and the blade base and tip j indices. The passage endwalls (j=1, j=jmax) are represented as solid surfaces (for a freestream outer boundary, see Standard Configuration #2). This is a relatively simple mesh to generate, and a number of grid generation schemes are available to construct this type of mesh. The advantage of this mesh system lies in the simplicity and consistency with the manner in which turbomachinery airfoils are defined and analyzed experimentally (streamline-like radial planes, constant axial planes). The disadvantages of this mesh system are that the airfoil leading and trailing edges are poorly defined because of the sheared mesh system, and grid points are often not used economically in the far field because of the requirement for mesh clustering along the periodic boundaries in the vicinity of the blade (although it is possible to construct meshes with 2-D mesh blocks in the far field to reduce this inefficiency).



ADPAC Standard Configuration #1
Single Passage Turbomachinery H−Grid

Configuration Information

Mesh Type	: Single Passage H−Grid
Number of Blocks	: 1
Flow Condition	: Steady
Applications	: Turbomachinery/Propfan Steady Flow Aerodynamics

Turbomachinery Cascade
Computational Domain
Solid wall or freestream outer boundary
i−j view
Can specify blade tip − tip clearance
j−k view
Airfoil
Can specify blade base − hub clearance
Airfoils
j
i
leading edge
trailing edge
j
k

Figure 6.1: Standard Configuration #1 Geometry and Multiple Block Mesh Structure

Standard Configuration #2

Title:	Single-passage ducted propfan H-grid
Mesh Type:	Two block H-grid
Number of Mesh Blocks:	2
Flow Type:	Steady
Geometry:	Ducted fan
Number of Blade Rows:	1
Mesh Generation Program:	*TIGG3D* used with *ROTGRID*, and others

Standard Configuration #2 consists of a pair of H-type mesh blocks discretizing a single blade passage of a ducted fan blade row, as shown in Figure 6.2. Both mesh blocks are spatially periodic, except in the vicinity of the blades themselves. The blades are represented as a circumferential displacement on the periodic surfaces (k=1 and k=kmax) of mesh block #1. A hub spinner boundary may be imposed as an embedded feature in mesh block #1. A blade tip clearance may be imposed by removing the blade displacement to satisfy spatial periodicity. The two mesh blocks are divided by the duct and a pair of mating surfaces which extend upstream and downstream from the duct leading and trailing edges. The duct itself is therefore represented as a gap between the two mesh blocks. The mesh points defining the mating surfaces are common to both mesh blocks. Inner boundaries on this mesh include the blade leading and trailing edge i indices, the duct leading and trailing edge i indices, the hub spinner leading and trailing edge i indices, and the blade tip j index. This mesh is best generated using the *TIGG3D* grid generation program in conjunction with the *ROTGRID* program to convert the single block *TIGG3D* mesh into the two block system described above, and to remove any duplicate mesh lines normally added by the *TIGG3D* program for ducted geometries. The advantage of this mesh system lies in the simplicity and consistency with the manner in which ducted fan airfoils are defined and analyzed experimentally (streamline-like radial planes, constant axial planes). The disadvantages of this mesh system are that the airfoil and duct leading and trailing edges are poorly defined because of the sheared mesh system, and grid points are often not used economically in the far field because of the requirement for mesh clustering along the periodic boundaries

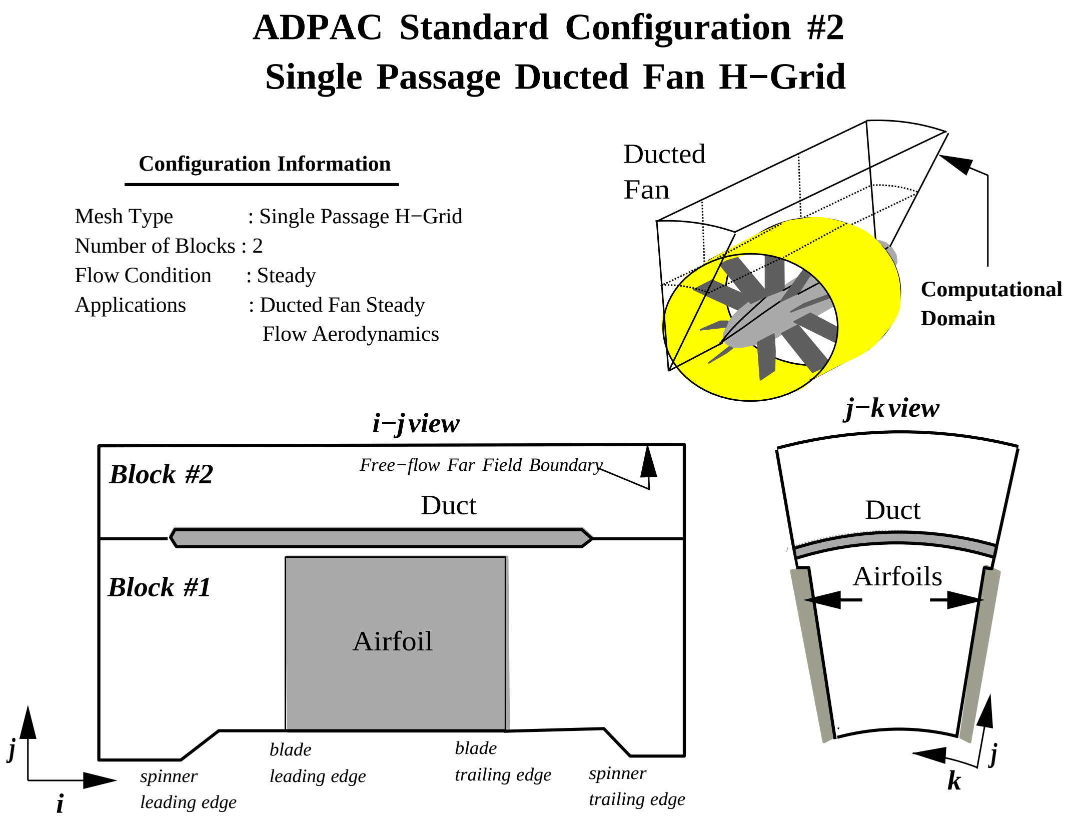


Figure 6.2: Standard Configuration #2 Geometry and Multiple Block Mesh Structure

in the vicinity of the blade and along radial planes in the vicinity of the duct (although it is possible to construct meshes with 2-D mesh blocks in the far field to reduce this inefficiency).

Standard Configuration #3

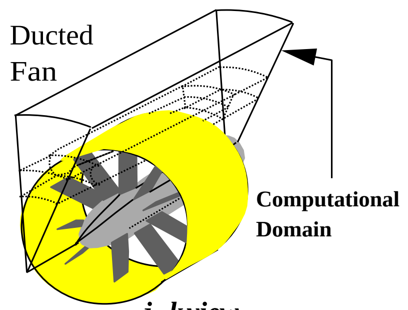
Title:	Single-passage ducted propfan C-grid
Mesh Type:	Five block combination C-H grid
Number of Mesh Blocks:	5
Flow Type:	Steady
Geometry:	Ducted fan
Number of Blade Rows:	1
Mesh Generation Program:	*CHGRIDV2* and others

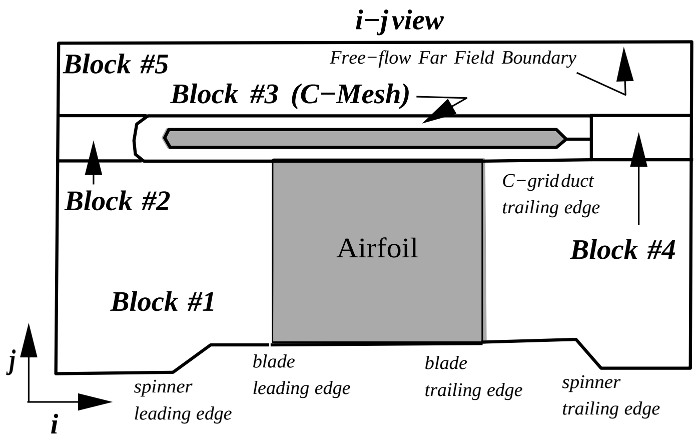
Standard Configuration #3 consists of a five block mesh system employing combined C and H type meshes, as shown in Figure 6.3, to discretize a single blade passage of a ducted fan blade row. All mesh blocks are spatially periodic, except in the vicinity of the blades themselves in block #1. The blades are represented as a circumferential displacement on the periodic surfaces (k=1 and k=kmax) of mesh block #1. A hub spinner boundary may be imposed as an embedded feature in mesh block #1. A blade tip clearance may be imposed by removing the blade displacement to satisfy spatial periodicity. The mesh blocks share common points along all of the mating surfaces between neighboring mesh blocks. Inner boundaries on this mesh include the blade leading and trailing edge i indices, the C-grid/H-grid mating i indices, the duct trailing edge i indices, and the hub spinner leading and trailing edge i indices. This mesh is best generated using the *CHGRIDV2* grid generation program. The advantage of this mesh system lies in the detailed representation of the duct leading edge permitted by the embedded C-grid, and the ability to efficiently cluster points about the duct surface and wake. The disadvantages of this mesh system are that the airfoil leading and trailing edges are poorly defined because of the sheared mesh system, and grid points are often not used economically in the far field because of the requirement for mesh clustering along the periodic boundaries in the vicinity of the blade and along radial planes in the vicinity of the duct. Another disadvantage of this mesh system is the complexity of the boundary specifications required to couple the mesh blocks, and the possibility of poor convergence as a result of the many inner domain block boundaries.

Single Passage Ducted Fan C−H Grid
ADPAC Standard Configuration #3

Configuration Information

Mesh Type	: Single Passage 5−Block C−H Grid System
Number of Blocks	: 5
Flow Condition	: Steady
Applications	: Ducted Fan Steady Flow Aerodynamics





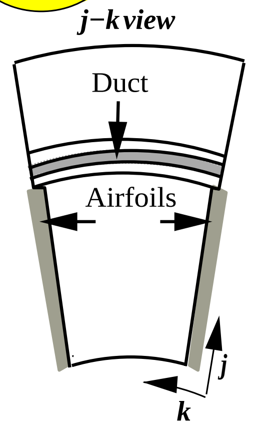


Figure 6.3: Standard Configuration #3 Geometry and Multiple Block Mesh Structure

Standard Configuration #4

Title:	Full rotor unducted propfan H-grid
Mesh Type:	Blade passage H-grid
Number of Mesh Blocks:	1*N (N=Number of Blades)
Flow Type:	Unsteady
Geometry:	Propeller, profan, unducted fan, compressor at angle of attack
Number of Blade Rows:	1
Mesh Generation Program:	*TIGG3D, CHGRIDV2, MULAC*, then use *ROTGRID*, or others

Standard Configuration #4 consists of multiple grid blocks discretizing a complete blade row of an unducted fan or internal turbomachinery blade row using a single H-type mesh block per blade passage. This mesh system is illustrated in Figure 6.4. Each blade passage mesh block is similar in form to the mesh systems in Standard Configuration #1, above. The individual blade passage mesh blocks are spatially periodic, except in the vicinity of the blades themselves. The blades are represented as a circumferential displacement on the periodic surfaces (k=1 and k=kmax) of each mesh block. Blade hub/tip clearances may be represented by removing the blade displacement in the clearance region above or below the blade. Inner boundaries on this mesh include the blade leading and trailing edge i indices, and the blade base and tip j indices. The passage endwalls (j=1, j=jmax) are represented as either solid surfaces or a freestream outer boundary. This is a relatively simple mesh to generate, and a number of grid generation schemes are available to construct this type of mesh for a single blade passage. The *ROTGRID* program can be used to create the duplicate blade passage mesh blocks for the full rotor geometry. Periodic surfaces between adjacent blade passages share common mesh points. The advantage of this mesh system lies in the simplicity and consistency with the manner in which turbomachinery airfoils are defined and analyzed experimentally (streamline-like radial planes, constant axial planes). The disadvantages of this mesh system are that the airfoil leading and trailing edges are poorly defined because of the sheared mesh system, and grid points are often not used economically in the far field

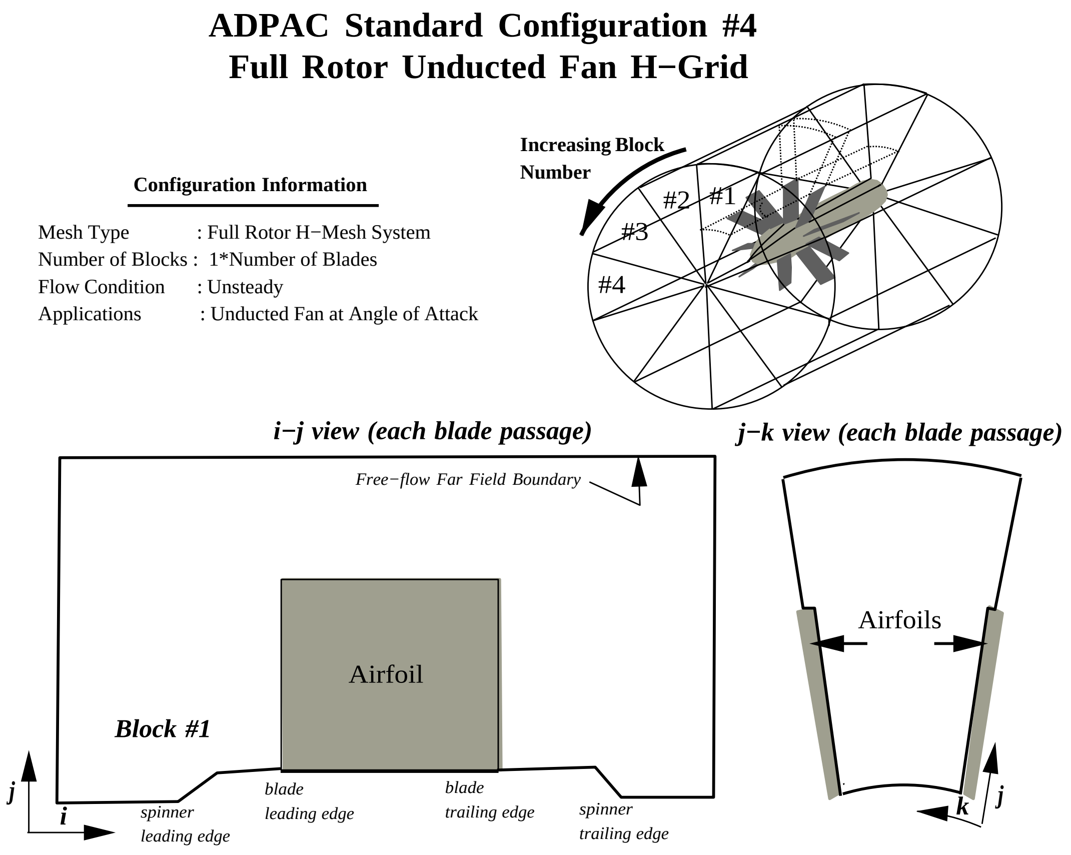

ADPAC Standard Configuration #4
Full Rotor Unducted Fan H−Grid
Configuration Information
Mesh Type : Full Rotor H−Mesh System
Number of Blocks : 1*Number of Blades
Flow Condition : Unsteady
Applications : Unducted Fan at Angle of Attack
Increasing Block Number
#1
#2
#3
#4
i−j view (each blade passage)
Free−flow Far Field Boundary
Airfoil
Block #1
j
i
spinner leading edge
blade leading edge
blade trailing edge
spinner trailing edge
j−k view (each blade passage)
Airfoils
k
j

Figure 6.4: Standard Configuration #4 Geometry and Multiple Block Mesh Structure

because of the requirement for mesh clustering along the periodic boundaries in the vicinity of the blade (although it is possible to construct meshes with 2-D mesh blocks in the far field to reduce this inefficiency).

Standard Configuration #5

Title:	Full rotor ducted propfan H-grid
Mesh Type:	2-Block H-grid per blade passage
Number of Mesh Blocks:	2*N (N=Number of Blades)
Flow Type:	Unsteady
Geometry:	Ducted Fan at Angle of Attack
Number of Blade Rows:	1
Mesh Generation Program:	*TIGG3D, CHGRIDV2, MULAC* used with *ROTGRID*, and others

Standard Configuration #5 consists of multiple grid blocks discretizing a complete blade row of a ducted fan using a pair of H-type mesh blocks per blade passage. Each blade passage consists of two mesh blocks in the same manner as the mesh system in Standard Configuration #2, above. The mesh block structure and numbering scheme is illustrated in Figure 6.5. The mesh blocks for a given blade passage are spatially periodic, except in the vicinity of the blades themselves. The blades are represented as a circumferential displacement on the periodic surfaces (k=1 and k=kmax) of the inner mesh block. A hub spinner boundary may be imposed as an embedded feature in the inner mesh block. A blade tip clearance may be imposed by removing the blade displacement to satisfy spatial periodicity. The two mesh blocks per blade passage are divided by the duct, and mating surfaces which extend upstream and downstream from the duct leading and trailing edges. The duct itself is therefore represented as a gap between the two mesh blocks in each passage. The mesh systems share common points along all of the mating surfaces between neighboring mesh blocks. Inner boundaries on this mesh include the blade leading and trailing edge i indices, the duct leading and trailing edge i indices, the hub spinner leading and trailing edge i indices, and the blade tip j index. A single blade passage mesh is best generated using the *TIGG3D* grid generation program in conjunction with the *ROTGRID* program to convert the single block *TIGG3D* mesh into the two block system per blade passage, and ultimately a full rotor mesh system. The advantage of this mesh system lies in the simplicity and consistency with the manner in which ducted fan airfoils are defined and analyzed experimentally

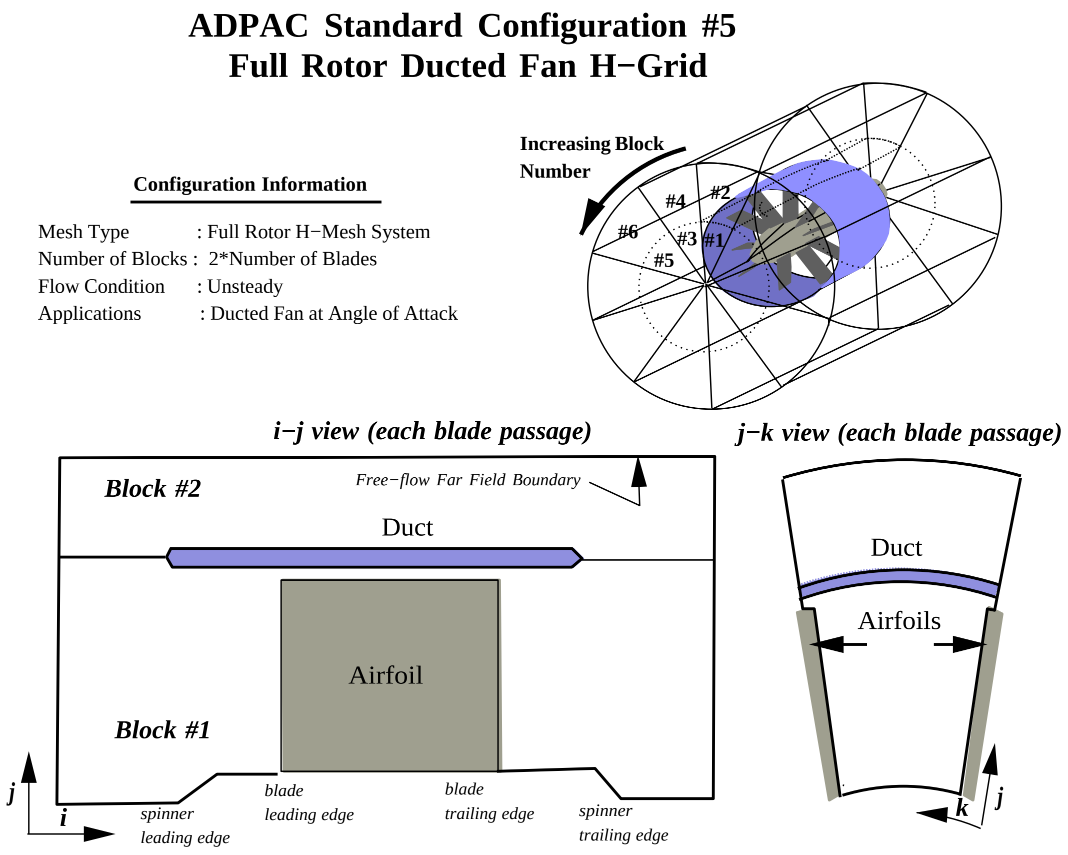

ADPAC Standard Configuration #5
Full Rotor Ducted Fan H−Grid
Configuration Information
Mesh Type : Full Rotor H−Mesh System
Number of Blocks : 2*Number of Blades
Flow Condition : Unsteady
Applications : Ducted Fan at Angle of Attack
Increasing Block Number
#1
#2
#3
#4
#5
#6
i−j view (each blade passage)
Block #2
Free−flow Far Field Boundary
Duct
Airfoil
Block #1
j
i
spinner leading edge
blade leading edge
blade trailing edge
spinner trailing edge
j−k view (each blade passage)
Duct
Airfoils
k
j

Figure 6.5: Standard Configuration #5 Geometry and Multiple Block Mesh Structure

(streamline-like radial planes, constant axial planes). The disadvantages of this mesh system are that the airfoil and duct leading and trailing edges are poorly defined because of the sheared mesh system, and grid points are often not used economically in the far field because of the requirement for mesh clustering along the periodic boundaries in the vicinity of the blade (although it is possible to construct meshes with 2-D mesh blocks in the far field to reduce this inefficiency).

Standard Configuration #6

Title:	Full Rotor Ducted Fan C-H Grid System
Mesh Type:	Five Block C-H Combination
Number of Mesh Blocks:	5*N (N=Number of Blades)
Flow Type:	Unsteady
Geometry:	Ducted Fan at Angle of Attack
Number of Blade Rows:	1
Mesh Generation Program:	*CHGRIDV2* used with *ROTGRID* and others

Standard Configuration #6 consists of multiple grid blocks discretizing a complete blade row of a ducted fan using a combination C-H mesh system per blade passage. Each blade passage consists of five mesh blocks in the same manner as the mesh system in Standard Configuration #3, above. The mesh block structure and numbering scheme is illustrated in Figure 6.6. The mesh blocks for a given blade passage are spatially periodic, except in the vicinity of the blades themselves. The blades are represented as a circumferential displacement on the periodic surfaces (k=1 and k=kmax) of the inner mesh block. A hub spinner boundary may be imposed as an embedded feature in the inner mesh block. The mesh systems share common points along all of the mating surfaces between neighboring mesh blocks. A single blade passage mesh system may be generated using the *CHGRIDV2* grid generation program. The full rotor mesh can then be constructed by using the *ROTGRID* program. The advantage of this mesh system lies in the detailed representation of the duct leading edge permitted by the embedded C-grid, and the ability to efficiently cluster points about the duct surface and wake. The disadvantages of this mesh system are that the airfoil leading and trailing edges are poorly defined because of the sheared mesh system, and grid points are often not used economically in the far field because of the requirement for mesh clustering along the periodic boundaries in the vicinity of the blade and along radial planes in the vicinity of the duct. (although it is possible to construct meshes with 2-D mesh blocks in the far field to reduce this inefficiency). Another disadvantage of this mesh system is the complexity of the boundary specifications required to couple the mesh blocks, and the possibility of poor convergence as a result of the many inner domain

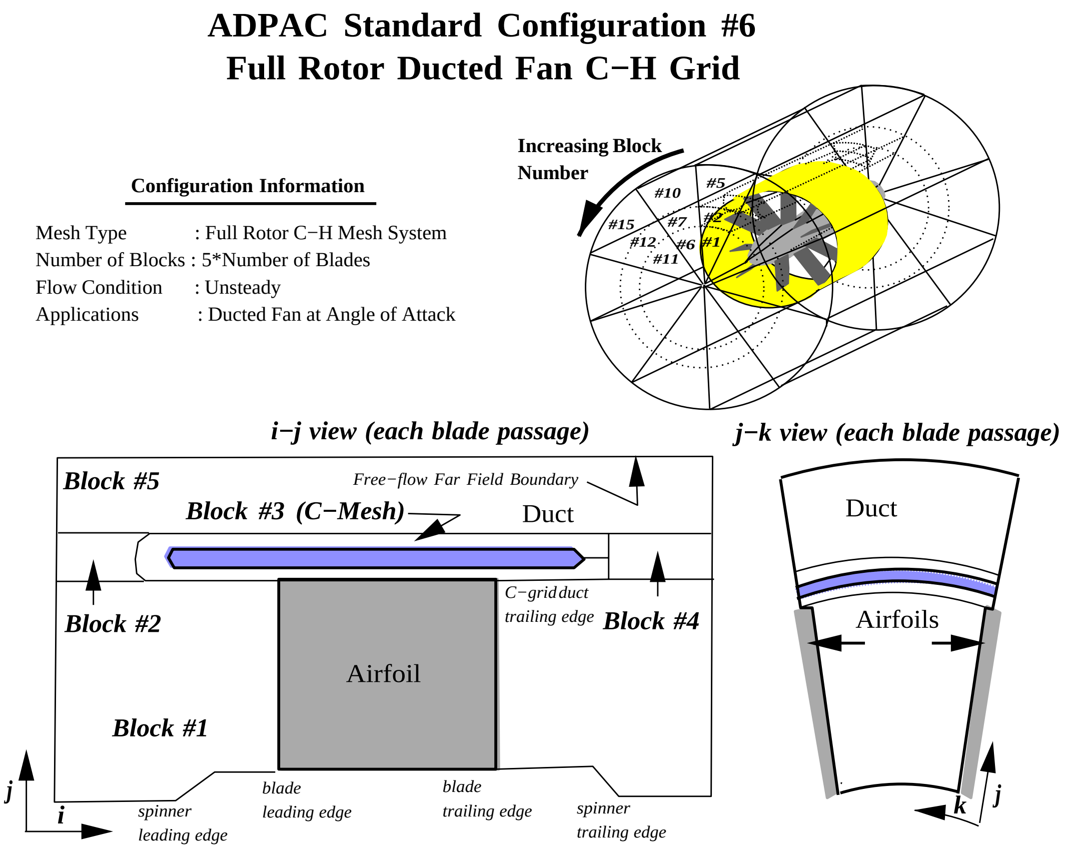

ADPAC Standard Configuration #6
Full Rotor Ducted Fan C−H Grid
Configuration Information
Mesh Type : Full Rotor C−H Mesh System
Number of Blocks : 5*Number of Blades
Flow Condition : Unsteady
Applications : Ducted Fan at Angle of Attack
Increasing Block Number
#10
#5
#15
#7
#2
#12
#6
#1
#11
i−j view (each blade passage)
Block #5
Free−flow Far Field Boundary
Block #3 (C−Mesh)
Duct
Block #2
C−grid duct trailing edge
Block #4
Airfoil
Block #1
j
i
spinner leading edge
blade leading edge
blade trailing edge
spinner trailing edge
j−k view (each blade passage)
Duct
Airfoils
k
j

Figure 6.6: Standard Configuration #6 Geometry and Multiple Block Mesh Structure

block boundaries.

<u>Standard Configuration #7</u>

Title:	Multiple Blade Row, Multiple Passage H-Grid
Mesh Type:	Single Block H-grid per Blade Passage
Flow Type:	Unsteady
Geometry:	Counterrotating propeller, profan, unducted fan, compressor
Number of Blade Rows:	Arbitrary
Mesh Generation Program:	*TIGG3D* and *ROTGRID*, or others

Standard Configuration #7 is a solution technique for analyzing the unsteady flow through a multiple blade row turbomachine. This configuration would be appropriate for the analysis of a counterrotating unducted fan, or for the internal flow in a multistage turbomachine. The mesh structure and boundary conditions are generated based on the time-dependent rotor-stator interaction solution approach discussed in Section 2.2. Several blade passages of each blade row are modeled, such that the total circumferential pitch for each blade row is constant. This implies that the number of blades in each row must be reducible to simple integer ratios (i.e. 3:4 for two blade rows, 3:4:6 for three blade rows, etc.), otherwise the number of passages which must be modeled for each blade row would be excessive. Each blade passage is discretized by a single H-type mesh block, similar to the mesh system described by Standard Configuration #1. The mesh block structure and numbering scheme is illustrated in Figure 6.7. The mesh block for a given blade passage is spatially periodic, except in the vicinity of the blades themselves. The blades are represented as a circumferential displacement on the periodic surfaces (k=1 and k=kmax) of each mesh block. At the interface between blade rows, the mesh surfaces share a common surface of revolution, and the axial and radial distribution of grid points in each mesh are identical at the interblade row interface (this reduces the interpolation of information between blade rows to the circumferential direction only).

ADPAC Standard Configuration #7
Multiple Blade Row/Multiple Passage Turbomachinery H−Grid

Configuration Information

Mesh Type	: Multiple Passage/Multiple Blade Row H−Grid
Number of Blocks	: Case Dependent Based on Number of Blade Rows and Number of Passages per Blade Row
Flow Condition	: Unsteady
Applications	: Turbomachinery Rotor/Stator Interaction

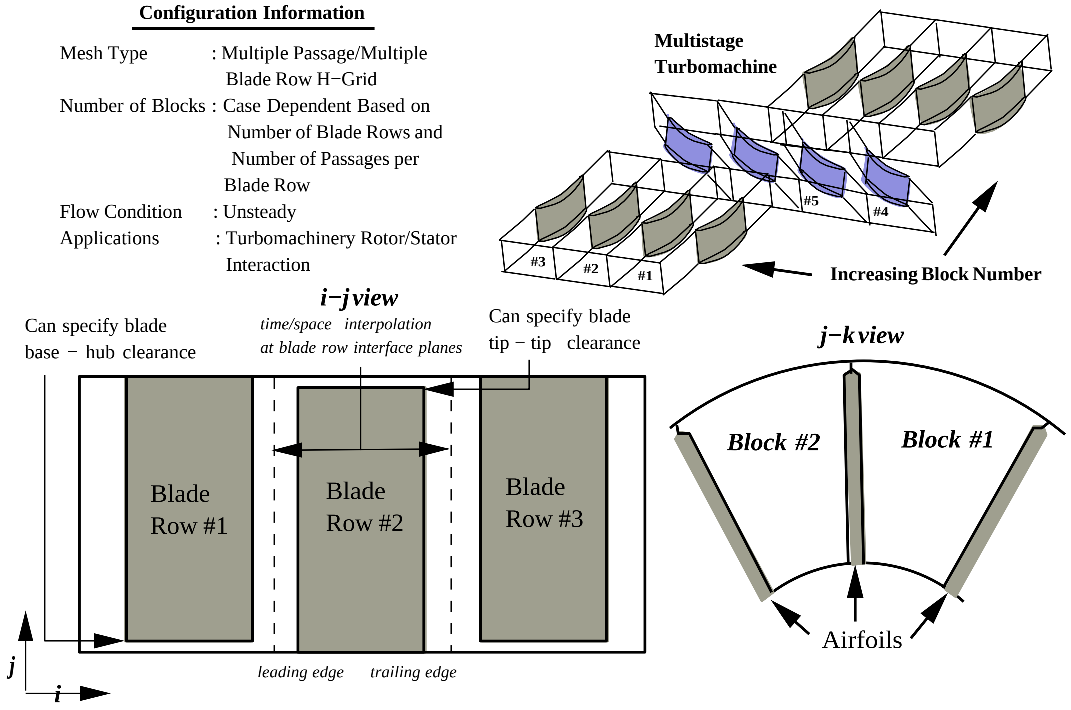
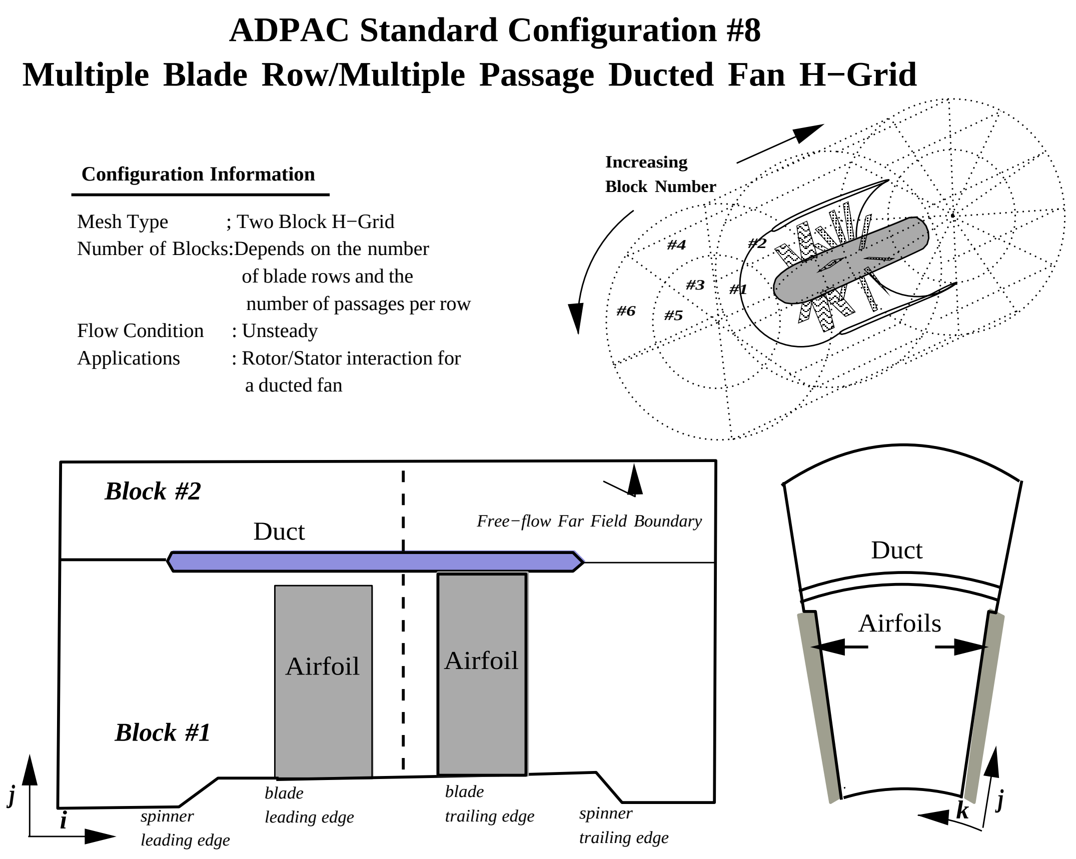


Figure 6.7: Standard Configuration #7 Geometry and Multiple Block Mesh Structure

Standard Configuration #8

Title:	Multiple Blade Row Multiple Passage Ducted Fan H-Grid
Mesh Type:	Two Block H-grid per Blade Passage
Flow Type:	Unsteady
Geometry:	Ducted fan
Number of Blade Rows:	Arbitrary
Mesh Generation Program:	*TIGG3D* followed by *ROTGRID*, or others

Standard Configuration #8 is a solution technique for analyzing the unsteady flow through a multiple blade ducted fan configuration, as shown in Figure 6.8. The mesh structure and boundary conditions are generated based on the time-dependent rotor-stator interaction solution approach discussed in Section 2.2. Several blade passages of each blade row are modeled, such that the total circumferential pitch for each blade row is constant. This implies that the number of blades in each row must be reducible to simple integer ratios (i.e. 3:4 for two blade rows, 3:4:6 for three blade rows, etc.), otherwise the number of passages which must be modeled for each blade row would be excessive. Each blade passage is discretized by a two H-type mesh blocks, similar to the mesh system described by Standard Configuration #2. The mesh block structure and numbering scheme is illustrated in Figure 6.8. The mesh blocks for a given blade passage are spatially periodic, except in the vicinity of the blades themselves. The blades are represented as a circumferential displacement on the periodic surfaces (k=1 and k=kmax) of the inner mesh block. For each blade passage, the two mesh blocks are divided by the duct and a pair of mating surfaces which extend upstream and downstream from the duct leading and trailing edges. The duct itself is therefore represented as a gap between the two mesh blocks. The mesh points defining the mating surfaces are common to both mesh blocks. The outer meshes are aligned circumferentially with the outer mesh distribution of the inner block. At the interface between blade rows, the mesh surfaces share a common surface of revolution, and the axial and radial distribution of grid points in each mesh are identical at the interblade row interface (this reduces the interpolation of information between blade rows to the circumferential



ADPAC Standard Configuration #8
Multiple Blade Row/Multiple Passage Ducted Fan H−Grid

Configuration Information

Mesh Type	; Two Block H−Grid
Number of Blocks	:Depends on the number of blade rows and the number of passages per row
Flow Condition	: Unsteady
Applications	: Rotor/Stator interaction for a ducted fan

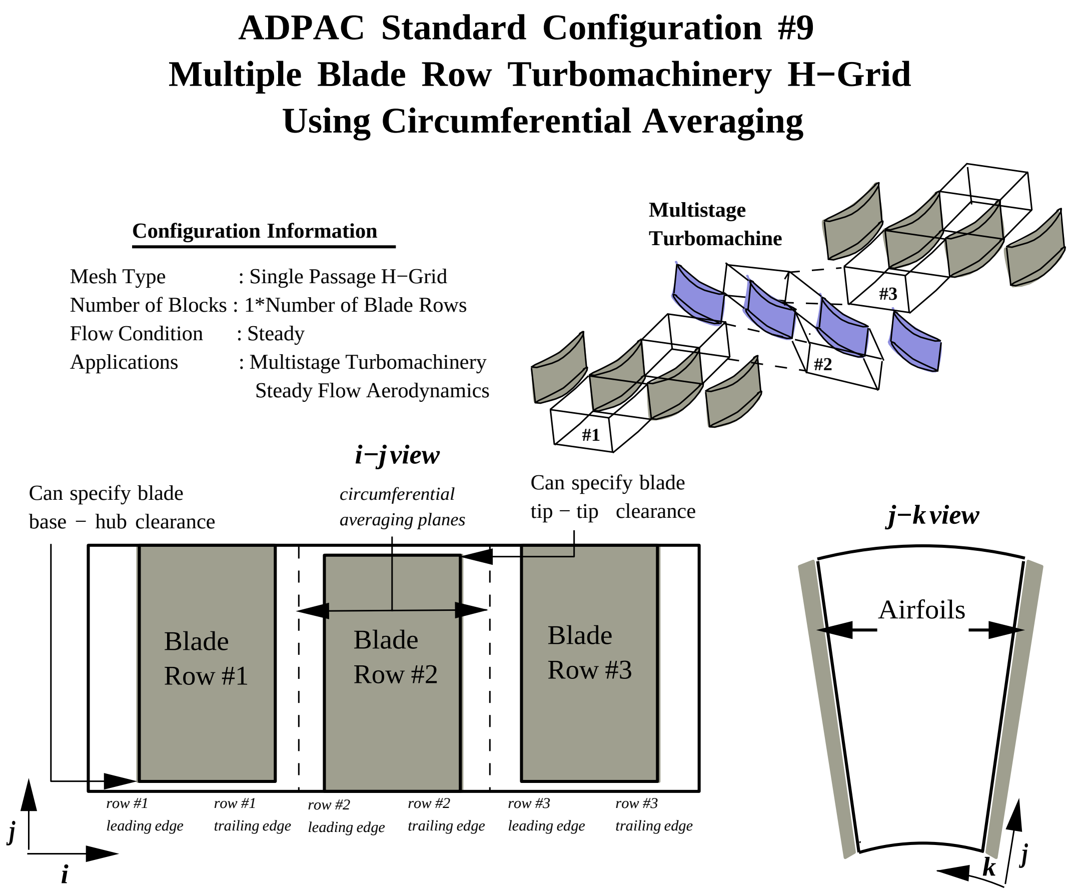
Figure 6.8: Standard Configuration #8 Geometry and Multiple Block Mesh Structure

direction only).

Standard Configuration #9

Title:	Multiple Blade Row Circumferential-Average H-Grid
Mesh Type:	Single Block H-grid per Blade Row
Flow Type:	Steady
Geometry:	Counterrotating propeller, profan, unducted fan, compressor
Number of Blade Rows:	Arbitrary
Mesh Generation Program:	*TIGG3D* and *ROTGRID*, or others

Standard Configuration #9 is a solution technique for analyzing the steady flow through a multiple blade row turbomachine. This configuration would be appropriate for the analysis of a counterrotating unducted fan, or for the internal flow in a multistage turbomachine. The mesh structure and boundary conditions are generated based on the mixing plane concept solution approach discussed in Section 2.2. A single blade passage in each blade row is modeled, with a surface of revolution mixing plane as the interface between blade rows. Each blade passage is discretized by a single H-type mesh block, similar to the mesh system described by Standard Configuration #1. The mesh block structure and numbering scheme is illustrated in Figure 6.9. The mesh block for a given blade passage is spatially periodic, except in the vicinity of the blades themselves. The blades are represented as a circumferential displacement on the periodic surfaces (k=1 and k=kmax) of each mesh block. At the interface between blade rows, the mesh surfaces share a common surface of revolution, and the axial and radial distribution of grid points in each mesh are identical at the interblade row interface (this simplifies the circumferential averaging operator).



ADPAC Standard Configuration #9
Multiple Blade Row Turbomachinery H−Grid
Using Circumferential Averaging

Configuration Information

Mesh Type	: Single Passage H−Grid
Number of Blocks	: 1*Number of Blade Rows
Flow Condition	: Steady
Applications	: Multistage Turbomachinery Steady Flow Aerodynamics

Figure 6.9: Standard Configuration #9 Geometry and Multiple Block Mesh Structure

<u>Standard Configuration #10</u>

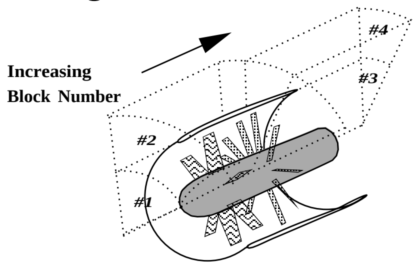
Title:	Multiple Blade Row Circumferential Average Ducted Fan H-Grid
Mesh Type:	Two Block H-grid per Blade Row
Flow Type:	Steady
Geometry:	Ducted fan
Number of Blade Rows:	Arbitrary
Mesh Generation Program:	*TIGG3D* followed by *ROTGRID*, or others

Standard Configuration #10 is a solution technique for analyzing the steady flow through a multiple blade ducted fan configuration, as shown in Figure 6.10. The mesh structure and boundary conditions are generated based on the mixing plane concept solution approach discussed in Section 2.2. A single blade passages of each blade row is modeled. Each blade passage is discretized by a two H-type mesh blocks, similar to the mesh system described by Standard Configuration #2. The mesh block structure and numbering scheme is illustrated in Figure 6.10. The mesh blocks for a given blade passage are spatially periodic, except in the vicinity of the blades themselves. The blades are represented as a circumferential displacement on the periodic surfaces (k=1 and k=kmax) of the inner mesh block. For each blade passage, the two mesh blocks are divided by the duct and a pair of mating surfaces which extend upstream and downstream from the duct leading and trailing edges. The duct itself is therefore represented as a gap between the two mesh blocks. The mesh points defining the mating surfaces are common to both mesh blocks. The outer meshes are aligned circumferentially with the outer mesh distribution of the inner block. At the interface between blade rows, the mesh surfaces share a common surface of revolution, and the axial and radial distribution of grid points in each mesh are identical at the mixing plane (this simplifies the circumferential averaging operator).

ADPAC Standard Configuration #10
Multiple Blade Row/Circumferential Average Ducted Fan H−Grid

Configuration Information

Mesh Type	; Two Block H−Grid
Number of Blocks	: 2*Number of Blade Rows
Flow Condition	: Steady
Applications	: Multistage Analysis of a Ducted Fan Using Circumferential Averaging



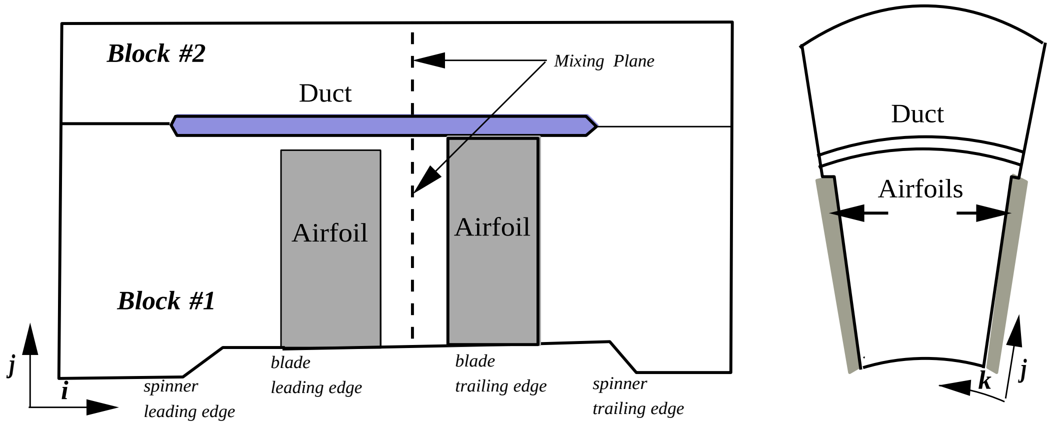


Figure 6.10: Standard Configuration #10 Geometry and Multiple Block Mesh Structure

Chapter 7

ROTGRID PROGRAM DESCRIPTION

The standard distribution for the *ADPAC08* program includes a program which aids the user in setting up mesh files for each of the standard configurations listed in Chapter 5. This program, referred to simply as *ROTGRID*, is an interactive, menu based program which queries the user for the specific information needed to properly configure a multiple-block mesh for the *ADPAC08* code for the given configuration. *ROTGRID* then goes about the task of collecting mesh coordinate data and writing a corresponding mesh file for the *ADPAC08* solution. *ROTGRID* is set up specifically for mesh files initially generated using the *TIGG3D* grid generation program. Under some circumstances, meshes generated by the *CHGRIDV2* program are also treated properly. If *ROTGRID* cannot handle a particular mesh system properly, the user can assemble an *ADPAC08* mesh from a collection of isolated mesh blocks by using the program *MAKEADGRID* described in Chapter 7.

7.1 Configuring *ROTGRID* Maximum Array Dimensions

Maximum array dimensions in the *ROTGRID* program are set by the FORTRAN PARAMETER statements listed in the file **parameter.inc** included with the *ROTGRID* source program. A sample **parameter.inc** file with the corresponding parameter descriptions would appear as:

```
C**********************************************************************
C
C  PARAMETER DESCRIPTION FILE:
C
C   imx --- > maximum number of grid elements in the i coordinate direction
C             for any given mesh block
C   jmx --- > maximum number of grid elements in the j coordinate direction
C             for any given mesh block
C   kmx --- > maximum number of grid elements in the k coordinate direction
C             for any given mesh block
C   nblks - > maximum number of grid blocks for the rotated grid
C   mrowmx --> maximum number of blade rows for the rotated grid
C
C---> Note: PARAMETER nrowmx must be >= 5.
C
        parameter(imx=225,jmx=50,kmx=91,nblks= 5,nrowmx=2)
```

7.2 Compiling the *ROTGRID* Program

The *ROTGRID* program source directory contains a UNIX-based Makefile facility to automate compilation for a number of machines. In the directory containing the FORTRAN source of the *SETUP* code, compilation is performed by executing the command:

make *option*

The **make** command is standard on UNIX systems and automatically interrogates the file *Makefile* for instructions on how to perform the compilation. The *option* argument may be any of the variables listed below:

No argument - same as *link* below.

link This is the standard UNIX system compilation. This option will deliver a working executable on most UNIX systems which support standard naming conventions (*f77* as the standard compiler, etc.). The compilation includes basic compiler optimization (f77 -O).

cray This option is utilized when compiling the *ROTGRID* code on a Cray computer.

aix This option is used when compiling the *ROTGRID* standard code on an IBM RS-6000 workstation running the AIX operating system.

7.3 Running the *ROTGRID* Program

Once the code has been compiled, change directories to the location where the case of interest has been stored. The *ROTGRID* program requires that the mesh has already been generated, as it will seek out the mesh file *case.mesh*, where *case* is the case name used in the file naming convention (see Section 3.5).

The *ROTGRID* program is invoked by issuing the command:

*path/***rotgrid**

where *path* is the relative or absolute pathname of the directory containing the *ROTGRID* executable file from the current local directory. For example, if the mesh file is in the directory

```
/usr/people/me/testcase
```

and the *ROTGRID* executable is in the directory

```
/usr/people/me/adpac/src/rotgrid
```

then the commands

```
cd /usr/people/me/testcase
/usr/people/me/adpac/src/rotgrid/rotgrid
```

would begin the *ROTGRID* program process.

Once the *ROTGRID* program is invoked, the user is asked to define the *case* name used in the file naming process (see Section 3.5). Following this, the user is asked to select the desired standard configuration. After the user selects one of the standard configurations, a series of specific questions pertaining to the requested configuration must be answered. Due to the large number of possibilities, it is not possible to explain all of the questions and answers in detail here. The responses to the program inquiries are intended to be fairly self-explanatory, and the user is urged to simply run the program for further details.

Chapter 8

MAKEADGRID PROGRAM DESCRIPTION

The standard distribution for the *ADPAC08* program includes a program called *MAKEADGRID* which aids the user in setting up a multiple-block mesh file from isolated unformatted mesh files. This program is useful for creating *ADPAC08* compatible multiple-block meshes from output from grid generation programs which are not supported by *ROTGRID* or which do not utilize the Scientific Database Library (SDBLIB). The *MAKEADGRID* program is an interactive program which queries the user for the number of blocks to be assembled for the final mesh, and then requests a file name for each of the individual mesh blocks. The user is then requested to name the final output file for the *ADPAC08* compatible multiple-block mesh. The individual mesh blocks are assembled in the order in which the mesh file names are specified, so care must be taken to order these names appropriately.

8.1 Configuring *MAKEADGRID* Maximum Array Dimensions

Maximum array dimensions in the *MAKEADGRID* program are set by the FORTRAN PARAMETER statements listed in the source file **makeadgrid.f** included with the standard distribution. The PARAMETER statement and the descriptions of the various parameter variables appear at the top of the file as:

```
C
C
C makeadgrid: This program assembles an ADPAC-compatible mesh file
C            from selected other unformatted PLOT3D mesh files
C
C
C  Set parameter size for max grid block to be read in
C
C   imax --- > maximum number of grid elements in the i coordinate direction
C              for any given mesh block
C   jmax --- > maximum number of grid elements in the j coordinate direction
C              for any given mesh block
C   kmax --- > maximum number of grid elements in the k coordinate direction
C              for any given mesh block
C   nnames - > maximum number of grid blocks for final mesh
C
        parameter(imax=251, jmax=82, kmax=53)
        parameter(nnames = 100 )
```

8.2 Compiling the *MAKEADGRID* Program

The *MAKEADGRID* program source directory contains a UNIX-based Makefile facility to automate compilation for a number of machines. In the directory containing the FORTRAN source of the *MAKEADGRID* code, compilation is performed by executing the command:

make *option*

The **make** command is standard on UNIX systems and automatically interrogates the file *Makefile* for instructions on how to perform the compilation. The *option* argument may be any of the variables listed below:

No argument - same as *link* below.

link This is the standard UNIX system compilation. This option will deliver a working executable on most UNIX systems which support standard naming conventions (*f77* as the standard compiler, etc.). The compilation includes basic compiler optimization (f77 -O).

cray This option is utilized when compiling the standard code on a Cray computer.

aix This option is used when compiling the standard code on an IBM RS-6000 workstation running the AIX operating system.

8.3 Running the *MAKEADGRID* Program

Once the code has been compiled, change directories to the location where the case of interest has been stored. The *MAKEADGRID* program requires that each individual mesh block for the final mesh be stored separately as a single-grid unformatted *PLOT3D* file .

The *MAKEADGRID* program is invoked by issuing the command:

*path/***makeadgrid**

where *path* is the relative or absolute pathname of the directory containing the *MAKEADGRID* executable file from the current local directory. For example, if the mesh file is in the directory

```
/usr/people/me/testcase
```

and the *MAKEADGRID* executable is in the directory

```
/usr/people/me/adpac/src/makeadgrid
```

then the commands

```
cd /usr/people/me/testcase
/usr/people/me/adpac/src/makeadgrid/makeadgrid
```

would begin the *MAKEADGRID* program process.

8.4 Sample Session Using the *MAKEADGRID* Program

A sample session using the *MAKEADGRID* program for the mesh illustrated in Figure 2.4 is given below. In this case, the mesh was originally generated using a proprietary mesh generation program, and hence, required some manipulation in order to construct the multiblock mesh for an *ADPAC08* solution. The mesh consists of 3 mesh blocks (the O-grid about the airfoil, and 2 H-grid caps upstream and downstream of the O-grid) named **block1.mesh**, **block2.mesh**, and **block3.mesh**. The *MAKEADGRID* session used to create the final mesh named **vbivane.mesh** is listed below. The user responses to the *MAKEADGRID* program are given in boldfaced type.

```
 ************************************************

  MAKEADGRID - construction program for
               creating ADPAC mesh files
               from selected PLOT3D unformatted
               mesh files.

 ************************************************



  Enter the number of blocks
```

3

```
  Enter the name of the            1 grid to process

  (Remember: each file must be unformatted PLOT3D style
```

block1.mesh

```
 Enter the name of the           2 grid to process

 (Remember: each file must be unformatted PLOT3D style
```

block2.mesh

```
 Enter the name of the           3 grid to process

 (Remember: each file must be unformatted PLOT3D style
```

block2.mesh

```
 Getting grid sizes and extra info from grid files

 Loop=           1 mg=           0 il,jl,kl=         129           33
          33
 Loop=           2 mg=           0 il,jl,kl=          17           33
          17
 Loop=           3 mg=           0 il,jl,kl=          17           33
          17

 Enter the file name for the final grid
```

vbivane.mesh

```
 Final grid data in file
vbivane.mesh
```

```
 Output file array size

 Loop =            1 -->          129          33          33
 Loop =            2 -->           17          33          17
 Loop =            3 -->           17          33          17
 Array sizes output to final file
 Reading Grid Data from file
block1.mesh
 il, jl, kl --->          129          33          33
 Output grid data to final file
 Reading Grid Data from file
block2.mesh
 il, jl, kl --->           17          33          17
 Output grid data to final file
 Reading Grid Data from file
block3.mesh
 il, jl, kl --->           17          33          17
 Output grid data to final file


 PROGRAM COMPLETED NORMALLY
```

Chapter 9

SETUP PROGRAM DESCRIPTION

The standard distribution for the *ADPAC08* program includes a program which aids the user in setting up an input file and a boundary data file for each of the standard configurations listed in Chapter 5. This program, referred to simply as *SETUP*, is an interactive, menu based program which queries the user for the specific information needed by the *ADPAC08* code for the given configuration, and then goes about the task of writing a corresponding input and boundary data file for an *ADPAC08* solution.

9.1 Configuring *SETUP* Maximum Array Dimensions

Maximum array dimensions in the *SETUP* program are set by the FORTRAN PARAMETER statements listed in the file **parameter.inc** included with the *SETUP* source program. A sample **parameter.inc** file would appear as:

```
parameter( nbmax  = 100 )
parameter( ninvar = 100 )
```

The PARAMETER variable NBMAX determines the maximum number of mesh blocks which can be manipulated using *SETUP*. The PARAMETER variable NINVAR determines the maximum number of input file variables which is permitted and need not be changed unless the *SETUP* program is modified to incorporate additional input file variables (which would then require a corresponding modification to the *ADPAC08* program).

9.2 Compiling the *SETUP* Program

The *SETUP* program source directory contains a UNIX-based Makefile facility to automate compilation for a number of machines. In the directory containing the FORTRAN source of the *SETUP* code, compilation is performed by executing the command:

make *option*

The **make** command is standard on UNIX systems and automatically interrogates the file *Makefile* for instructions on how to perform the compilation. The *option* argument may be any of the variables listed below:

No argument - same as *link* below.

link This is the standard UNIX system compilation. This option will deliver a working executable on most UNIX systems which support standard naming conventions (*f77* as the standard compiler, etc.). The compilation includes basic compiler optimization (f77 -O).

cray This option is utilized when compiling the *SETUP* code on a Cray computer.

aix This option is used when compiling the *SETUP* code on an IBM RS-6000 workstation running the AIX operating system.

9.3 Running the *SETUP* Program

Once the code has been compiled, change directories to the location where the case of interest has been stored. The *SETUP* program requires that the mesh has already been generated, as it will seek out the mesh file *case.mesh*, where *case* is the case name used in the file naming convention (see Section 3.5).

The *SETUP* program is invoked by issuing the command:

*path/***setup**

where *path* is the relative or absolute pathname of the directory containing the *SETUP* executable file from the current local directory. For example, if the mesh file is in the directory

```
/usr/people/me/testcase
```

and the *SETUP* executable is in the directory

```
/usr/people/me/adpac/src/setup
```

then the commands

```
cd /usr/people/me/testcase
/usr/people/me/adpac/src/setup/setup
```

would begin the *SETUP* program process.

Once the *SETUP* program is invoked, the user is first requested to enter the *case* name used in the *ADPAC08* file naming convention (see Section 3.5). Following a series of questions used to define the flow condition, the user indicates which standard configuration is desired, and the *SETUP* program proceeds from there to construct a standard input file and a boundary data file for the given configuration based on the user input. The grid information indicated in the appropriate standard configuration illustration (Figures 6.1-6.10) should be available, as well as the desired flow conditions. The interactive questions and responses for the *SETUP* program are intended to be self-explanatory, and the user is urged to simply run the program for further details.

Chapter 10

ADPAC08 INTERACTIVE GRAPHICS DISPLAY

The *ADPAC08* program is equipped with an option which permits real time interactive graphics display of flow data in the form of colored contours or velocity vectors on geometries represented by wiremesh grid surfaces. The interactive graphics are based largely on routines generated from the *PLOT3D* visualization program, and many of the features of this option should be familiar to anyone who has used *PLOT3D*. All interactive graphics must be displayed on a Silicon Graphics workstation, IRIX Operating System 4.0.1 or above. The graphics display can be operated on a single computing platform, or can be directed across a network for specific computer hardware configurations. Thus, it is possible to have a job running remotely on a Cray computer, with interactive graphics displayed locally on a network-connected Silicon Graphics workstation. When operating across a network which involves a non-Silicon Graphics computer, the communication program *AGTPLT-LCL* must be running on the local display device in order to capture the graphics commands issued by the remote compute server (details on *AGTPLT-LCL* are given below). A graphic illustrating the possible graphics display operating modes is given in Figure 10.1. It should be mentioned that the interactive graphics display was not a contracted feature, and was actually developed to aid in debugging the multiple block code. The description of this feature is included in this manual for completeness, but the user should be cautioned due to the immature nature of this portion of the code. It is also likely that the graphics option may not port correctly to future releases of the IRIX operating system, and again, the user is cautioned concerning the use of this feature.

10.1 Setting up the Program

The first step in producing the real time interactive graphics display is to correctly compile the code to include the graphics libraries. This is accomplished by utilizing the appropriate option in the *ADPAC08* Makefile command (see Section 3.4). The valid graphics options include *graphics, pfagraphics, craygraphics, aixgraphics*, and *craygraphdbx*. These options incorporate various levels of the included graphics libraries for execution on various machines (again, see Section 3.4 for specific Makefile details).

Once the code has been correctly compiled to include the graphics libraries, several input parameters must be correctly initiated to engage the graphics subroutines during the execution of the code. The input keyword **FGRAFIX** must have a value of 1.0 to initiate any graphics instructions. The keyword **FGRAFINT** determines the number of time-marching iterations between graphics window updates. The keyword **FIMGSAV** is a trigger (0.0 - off, 1.0 - on) which determines whether periodic image capturing is enabled, and the keyword **FIMGINT** determines the number of time-marching iterations between image captures. Additional details concerning these input file keywords are available in Section 3.6.

10.2 Graphics Window Operation

Once the graphics window has been initiated on the local display, and the initial data has been plotted, the program continues and the graphics display data are updated every **FGRAFINT** iterations. This process will continue until the program terminates, or until the user interrupts the process by pressing the left mouse button once with input focus directed to the graphics display window. A short time later, (the delay may be quite long for a network which is burdened), the graphics display will freeze, and the computational portions of the program will be suspended in order to permit the

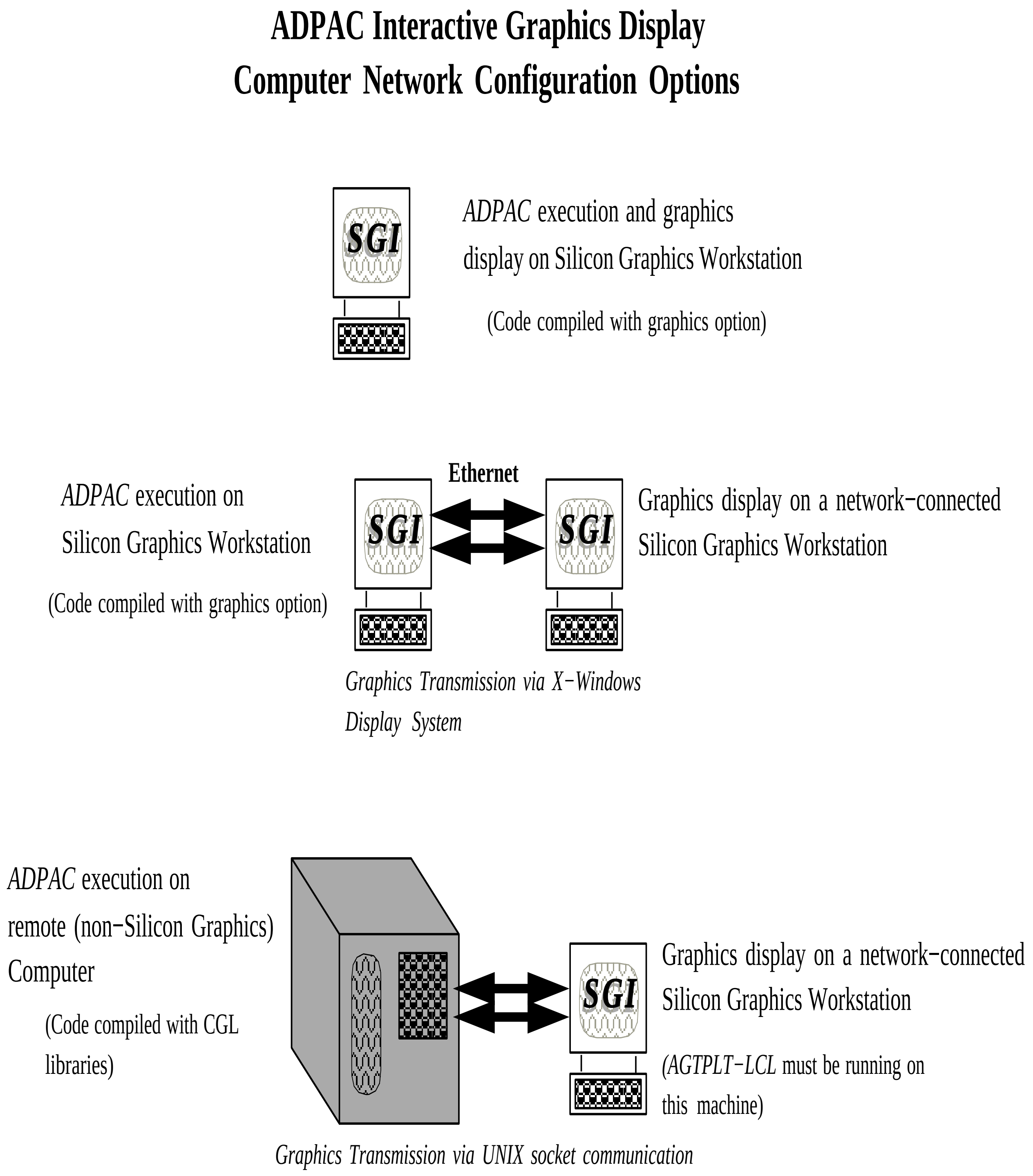

ADPAC Interactive Graphics Display
Computer Network Configuration Options
SGI
ADPAC execution and graphics display on Silicon Graphics Workstation
(Code compiled with graphics option)
Ethernet
ADPAC execution on Silicon Graphics Workstation
(Code compiled with graphics option)
Graphics display on a network-connected Silicon Graphics Workstation
Graphics Transmission via X-Windows Display System
ADPAC execution on remote (non-Silicon Graphics) Computer
(Code compiled with CGL libraries)
Graphics display on a network-connected Silicon Graphics Workstation
(AGTPLT-LCL must be running on this machine)
Graphics Transmission via UNIX socket communication

Figure 10.1: *ADPAC08* Interactive Graphics Display Network Configuration Options

user to interactively translate, rotate, or scale the graphics image to their liking. When the display has been frozen, the viewpoint of the display may be altered by one of several mouse controls. The left mouse button controls rotation, the right mouse button controls translation, and the middle mouse button controls scaling (zoom in, zoom out). The controlling mouse movements are illustrated in Figure 10.2. The mouse-directed viewpoint controls are identical to those used in *PLOT3D* [11]. Once the viewpoint has been altered, program control is returned to *ADPAC08* by hitting the ENTER key on the keyboard with input focus directed to the graphics window. At this point, the code will then return to the process of performing time-marching iterations, with periodic updating of the graphics screen.

It is also possible for the user to change the plotting function by entering any one of the following characters with input focus directed to the graphics window at any time during the process:

Key	Result
p	Set flow function to pressure contours
2	Set flow function to velocity vectors

The surfaces plotted by the interactive graphics display is currently hard-wired in the code. A wiremesh representation and the corresponding surface contours are generated for the i=1, j=1, and k=1 mesh surfaces. This restriction could be removed in future developments.

10.3 AGTPLT-LCL Program Description

The program *AGTPLT-LCL* is the receiving program for local graphics display of an *ADPAC08* job running on a remote, network-connected computing platform. The *AGTPLT-LCL* program is a modified version of the NASA-AMES developed *PLOT3D-LCL* program. This program can only be run on a Silicon Graphics Workstation running at level 4.0.1 (or above) of the IRIX operating system. As such, compilation of the *AGTPLT-LCL* program has no options, and is performed simply by executing the command **make** in the *AGTPLT-LCL* source directory. Once initiated, the *AGTPLT-LCL*

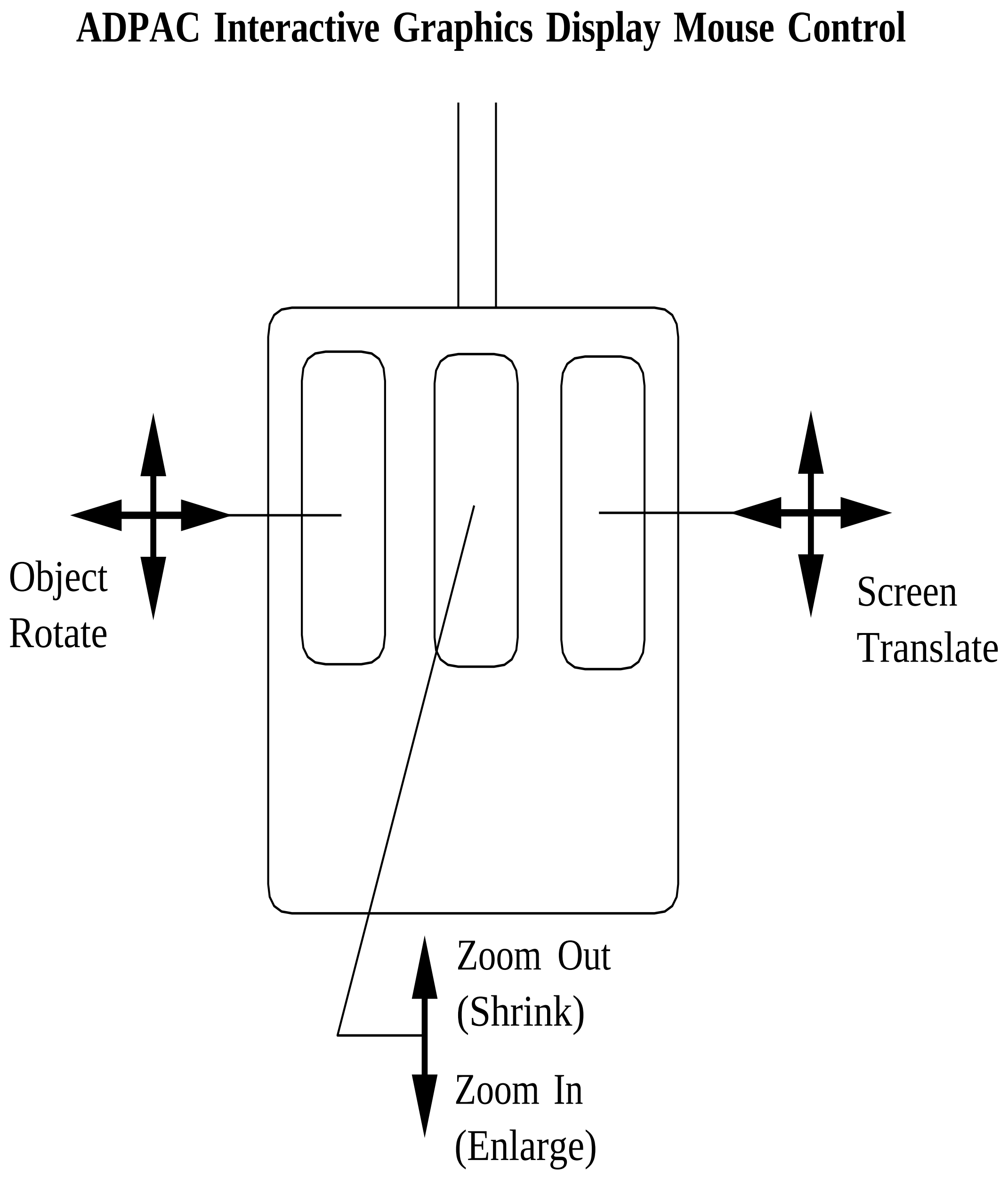


Figure 10.2: *ADPAC08* Interactive Graphics Display Mouse Control

program waits for an outside process from *ADPAC08* to communicate with the local workstation, and graphics commands received from the remote job are displayed locally.

An important consideration in setting up a remote calculation with local graphics display using *AGTPLT-LCL* is the manner in which the local display is defined in the calculation. The CGL libraries used to permit the network graphics instructions require an internet network address in order to properly transmit the graphics commands to the correct destination. This definition should be provided in the standard input file following the normal keyword parameters (see Section 3.6 for a sample file and keyword definitions). At the end of the standard input keyword data, the user should use an **ENDINPUT** statement to terminate the normal input stream. The **ENDINPUT** statement should then be followed by two blank lines, and then a line containing the destination network address of the local Silicon Graphics display device. This specification will ultimately be read by the CGL libraries in setting up the network connection.

The procedure to set up this network-connected graphics display option would be to start the job on the remote machine, and then immediately start the *AGTPLT-LCL* program on the local display. As long as the correct network address has been entered in the *case*.**input** file, then the remote program should begin communicating with the *AGTPLT-LCL* program, and the local graphics window will begin displaying the graphics instructions specified by the remote computing program.

Chapter 11

ADPAC08 TOOL PROGRAMS DESCRIPTION

The standard distribution for the *ADPAC08* program includes a number of tool programs designed to assist in examining and manipulating data generated for an *ADPAC08* solution. Although running these programs is generally self-explanatory, a brief description is provided below to outline the function of each tool program.

11.1 *ADPERF* Tool Program Description

The *ADPERF* tool program was designed to provide a simple post processing program for computing overall integrated thrust and power coefficients for unducted fan (propeller) calculations based on a simple H-type mesh discretization strategy (see Standard Configuration #1 in Chapter 5). Upon execution, the *ADPERF* program asks the user to input the name of the *ADPAC08* mesh and restart files for the run of interest. The *ADPERF* program then opens and reads both files, and attempts to estimate the number of blades in the propeller which the user must then verify (presumably the mesh represents only a single blade passage of the overall geometry). Following this, the *ADPERF* program asks for the value of the *ADPAC08* nondimensional parameters RHO0 and OMEGA. These values are identified in the *ADPAC08* output file under the following heading:

```
non-dimensional initial values calculated as:
********************************************
     rho0   ( initial density              ) =       .8498   <-----------
     u0     ( initial axial velocity       ) =       .6643
     v0     ( initial radial velocity      ) =       .0000
     w0     ( initial theta velocity       ) =       .0000
     ei0    ( initial internal energy      ) =      2.5630
     h0     ( initial enthalpy             ) =      3.5000
     p0     ( initial pressure             ) =       .7962
     t0     ( initial temperature          ) =       .9370
     dmu0   ( initial viscosity            ) =       .0000
     omega  ( rotational speed             ) =      -.0096   <-----------
```

The final parameter to be entered is the propeller diameter in grid units (if the mesh is in feet, enter the propeller diameter in feet). Following this, the *ADPERF* program will compute the propeller power and thrust coefficients based on blade static pressure loading.

11.2 *ADSTAT* Tool Program Description

The *ADSTAT* tool program was designed to provide statistical information about a mesh or flow (PLOT3D output) file from an *ADPAC08* run. Upon execution, the *ADSTAT* program asks the user to select whether information about a mesh file (m) or flow file (f) is desired. In either case, the user is then asked to input the appropriate mesh or flow file name. If the mesh file option is selected, the *ADSTAT* program open the mesh file and reports the number of mesh blocks contained within the file, as well as the individual mesh block sizes. The *ADSTAT* program also computes the maximum allowable number of multigrid levels (based on mesh size alone) which can be used for an *ADPAC08* run. In addition, the *ADSTAT* program computes and reports the minimum required *ADPAC08* array size parameters for all allowable number of multigrid levels. This capability is the most useful feature of the *ADSTAT* program. If the flow file option is selected, in addition to the above, the extra flow file data (standard in the PLOT3D file format) is also reported for each block (normally this includes the Mach number, angle of attack, reynolds number, and time).

11.3 *AOA2AXI* Tool Program Description

The *AOA2AXI* tool program was designed to compute an axisymmetric average of a 3-D cylindrical coordinate system solution. The program is restricted to H-type meshes similar to standard configurations #1-3 and in Chapter 5 which possess uniform axisymmetric projections on each mesh plane in the circumferential direction (this simplifies the averaging process). When running *AOA2AXI*, the user is requested to enter the 3-D mesh and flow (PLOT3D format) file names. Then, the user is offered the option of redimensionalizing the data, and finally, the user is requested to enter the 2-D axisymmetric mesh and flow (PLOT3D format) file names. The *AOA2AXI* code computes the axisymmetric average of the 3-D mesh and flow file data and stores the result in the 2-D axisymmetric mesh and flow files. These data may then be used with **PLOT3D** and other graphics visualization tools to examine the axisymmetric average of the 3-D solution.

11.4 *PLOT3D* Tool Programs Description

A number of tool programs originally generated for the *PLOT3D* program are included with the *ADPAC08* distribution because of their usefulness in manipulating *ADPAC08* mesh and flow (PLOT3D output format) files. A brief description of these codes is given below. It should be noted that most of these programs are designed to deal with unformatted files, rather than the *ADPAC08* standard binary format. Fortunately, the *PLOT3D* program can be used to convert from *ADPAC08* binary to unformatted, and the *MAKEADGRID* program may be used to convert from unformatted format to *ADPAC08* binary.

- **CHOPQ** Cut a subset out of a (3D, single grid) PLOT3D Q file and write it out as a new Q file.
- **CHOPX** Cut a subset out of a (3D, single grid) PLOT3D XYZ file and write it out as a new XYZ file.
- **CHOPXB** Cut a subset out of a (3D, single grid) PLOT3D XYZ+IBLANK file and write it out as a new XYZ+IBLANK file.
- **COMBINEQ** Combine several (3D, single grid) PLOT3D Q files into a new multiple grid Q file.
- **COMBINEX** Combine several (3D, single grid) PLOT3D XYZ files into a new multiple grid XYZ file.
- **COMBINEXB** Combine several (3D, single grid) PLOT3D XYZ+IBLANK files into a new multiple grid XYZ+IBLANK file.
- **IJK** Generate a (3D, single grid) PLOT3D XYZ file which is simply the computational grid, i.e. (x,y,z)=(i,j,k). Good for looking at flow quantities in the computational domain.
- **INT3D** Interpolate a (3D, single grid) PLOT3D Q file from one grid onto another. Old and new XYZ files may have IBLANK. Various options available on what to do if a new grid point isn't found within the old grid. Uses trilinear interpolation.
- **MIRRORQ** Flip a (3D, single grid) PLOT3D Q file about the x-, y-, or z-axis.
- **MIRRORX** Flip a (3D, single grid) PLOT3D XYZ file about the x-, y-, or z-axis.
- **PROPER2D** Perform 2D grid line crossing check on a (2D, single grid) PLOT3D XYZ file.
- **PROPER3D** Perform tetrahedron decomposition cell volume check on a (3D, single grid) PLOT3D XYZ file.
- **PROPER3DN** Perform tetrahedron decomposition grid crossing check on a (3D, single grid) PLOT3D XYZ file.

- **REFINEX** Generate a new (3D, single grid) PLOT3D XYZ file which is an integer refinement of an existing grid file. Uses parametric cubic interpolation.
- **ROTATEX** Rotate a (3D, single grid) PLOT3D XYZ file about the x-, y-, or z-axis.
- **SCALEX** Scale a (3D, single grid) PLOT3D XYZ file.
- **SCALEX** Scale a (3D, single grid) PLOT3D XYZ+IBLANK file.
- **SPLITQ** Split a (3D) multiple grid PLOT3D Q file into separate single grid Q files. Can skip grids if desired.
- **SPLITX** Split a (3D) multiple grid PLOT3D XYZ file into separate single grid XYZ files. Can skip grids if desired.
- **SPLITXB** Split a (3D) multiple grid PLOT3D XYZ+IBLANK file into separate single grid XYZ+IBLANK files. Can skip grids if desired.
- **TRANSLATEX** Translate a (3D, single grid) PLOT3D XYZ file.
- **TRANSLATEXB** Translate a (3D, single grid) PLOT3D XYZ+IBLANK file.

The UNIX *make* command may be used to compile and link the PLOT3D tools as follows:

```
2xxx/3xxx: make -f Makefile.i2
CRAY 2: make -f Makefile.c2
VAX/VMS: @MAKEFILE.VMS
```

Bibliography

[1] Hall, E. J. and Delaney, R. A., "Investigation of Advanced Counterrotation Blade Configuration Concepts for High Speed Turboprop Systems: Task V - Unsteady Counterrotation Ducted Propfan Analysis, Final Report", NASA CR 187126, NASA Contract NAS3-25270, 1992.

[2] Hall, E. J., Delaney, R. A., and Bettner, J. L., "Investigation of Advanced Counterrotation Blade Configuration Concepts for High Speed Turboprop Systems: Task II - Unsteady Ducted Propfan Analysis, Final Report", NASA CR 187106, NASA Contract NAS3-25270, 1991.

[3] Hall, E. J., Delaney, R. A., and Bettner, J. L., "Investigation of Advanced Counterrotation Blade Configuration Concepts for High Speed Turboprop Systems: Task II - Unsteady Ducted Propfan Analysis, Computer Program Users Manual", NASA CR 187105, NASA Contract NAS3-25270, 1991.

[4] Rao, K. V., and Delaney, R. A., 1990, "Investigation of Unsteady Flow Through a Transonic Turbine Stage: Part I- Analysis", AIAA Paper 90-2408.

[5] Jorgenson, P. C. E., and Chima, R. V., "An Unconditionally Stable Runge-Kutta Method for Unsteady Flows," AIAA Paper 89-0205, 1989.

[6] Adamczyk, J. J., "Model Equation for Simulating Flows in Multistage Turbomachinery," ASME Paper 85-GT-226, 1985.

[7] Dawes, W.N., "Multi-Blade Row Navier-Stokes Simulations of Fan Bypass Configurations", ASME Paper 91-GT-148, 1991.

[8] Crook, A. J., and Delaney, R. A., "Investigation of Advanced Counterrotation Blade Configuration Concepts for High Speed Turboprop Systems: Task IV - Advanced Fan Section Analysis, Final Report", NASA CR 187128, NASA Contract NAS3-25270, 1992.

[9] Rai, M. M., "Unsteady Three-Dimensional Navier-Stokes Simulations of Turbine Rotor-Stator Interaction," AIAA Paper 87-2058, 1987.

[10] Whipple, D., "BDX-Binary Data Exchange Preliminary Information", NASA-Lewis Research Center, 1989.

[11] Walatka, P. P., and Buning, P. G., "PLOT3D User's Manual,", rough draft for NASA TM, 1988.

[12] Plessel, Todd, "SURF User's Guide,", NASA Ames Research Center, 1988.

[13] Walatka, P. P., and Buning, P. G., "FAST", NASA Ames Research Center, 1990.

[14] Hall, E. J., Delaney, R. A., and Bettner, J. L., "Investigation of Advanced Counterrotation Blade Configuration Concepts for High Speed Turboprop Systems: Task I - Ducted Propfan Analysis", NASA CR 185217, NASA Contract NAS3-25270, 1990.

[15] Crook, A. J., and Delaney, R. A., "Investigation of Advanced Counterrotation Blade Configuration Concepts for High Speed Turboprop Systems: Task III - Advanced Fan Grid Generation", NASA CR 187129, NASA Contract NAS3-25270, 1991.

[16] Arnone, Andrea, *Notes on the Use of the TRAF Codes.* Department of Energy Engineering, University of Florence, and Institute for Computational Mechanics in Propulsion, NASA Lewis Research Center, November, 1992.

[17] Hall, E. J., Topp, D. A., Heidegger, N. J., and Delaney, R. A., "Investigation of Advanced Counterrotation Blade Configuration Concepts for High Speed Turboprop Systems: Task VII - Unsteady Inlet Distortion Analysis", to be published.

[18] Quealy, A., Cole, G., and Blech, R., "Portable Programming on Parallel/Network Computers Using the Application Portable Parallel Library (APPL)", NASA TM-106238, July, 1993.

Appendix A

ADPAC08 DISTRIBUTION AND DEMONSTRATION INSTRUCTIONS

A.1 Introduction

This appendix describes the commands necessary to extract the source code and demo files from the *ADPAC08* standard distribution and run a complete test case for a ducted fan employing multiple blade rows. The standard *ADPAC08* distribution is a compressed *tar* file which can be decoded into the various parts by a sequence of commands on any standard UNIX system. The sequence listed below is intended to guide the user through the setup from the standard distribution up to and including a complete demonstration of a calculation for a ducted propfan employing multiple blade rows. The command sequence listed below should work on most systems employing the UNIX operating system. Since portions of this process are inherently machine-dependent, the exact commands listed here are for a Silicon Graphics Workstation running the IRIX Operating System, Revision 4.0.1. Alternate commands will be listed when a significant machine dependence exists.

A.2 Extracting the Source Files

The *ADPAC08* programs are distributed as a compressed *tar* file named

adpac08.tar.Z

This tar file requires roughly 22.0 megabytes of disk space. It should be possible to extract and run the code on any standard UNIX system from this distribution file. The first step necessary to extract the *ADPAC08* programs is to uncompress the *tar* file with the command:

uncompress adpac08.tar.Z

This operation essentially replaces the compressed file *adpac08.tar.Z* with an uncompressed file *adpac08.tar*. The uncompressed *tar* file requires approximately 41.0 megabytes of disk space.

The next step is to extract the individual files and directories from the *adpac08.tar* file. The **tar** command will create a subdirectory named *adpac08* in the current directory, so it is up to the user to move the *adpac08.tar* file to a suitable initial directory before extracting the embedded subdirectories. Once the *tar* file is properly placed, the *ADPAC08* distribution may be extracted with the command

tar xvof adpac08.tar

(On some systems **tar xvf adpac08.tar** may be sufficient.) Execution of the UNIX list command **ls -l** will verify that the *adpac08* directory has been created. The complete extraction process will require about 90.0 Megabytes of disk space (to hold the *adpac08.tar* file and the extracted contents).

The *uncompress* and *tar* steps can be combined in a single operation on most *UNIX* systems by issuing the command

```
zcat adpac08.tar.Z | tar xvf
```

This combined operation conserves overall disk space requirements during the extraction process.

A.3 Compiling the Source Code

After extracting the source files, the user is naturally interested in compiling the source files for execution. A UNIX-compatible *Make* facility is provided for each of the *ADPAC08* programs. The *Makefile* which governs the compilation process is necessarily machine-dependent and requires that the user select from one of a number of preconfigured systems. The *Make* command is fully described in Section 3.4. If no option is specified in the *make* command, then the standard UNIX compilation is performed.

In order to begin the compilation, it is first necessary to enter the *adpac08* directory with the command:

```
cd adpac08
```

At this point, several files and directories will be available. By entering the UNIX command **ls -l**, a listing of the individual directories can be obtained. The output of the **ls** command will look something like:

```
README demo/     manual/     report/     src/
```

A description of each of these listings is given below:

README	This file is a general description of the contents of the directory.
demo	This directory contains several geometry and flow input files for generating sample runs of the *ADPAC08* codes.
manual	This directory contains the *LaTeX* source code for this manual. If *LaTeX* is installed on your system, it is possible to reproduce

this document (excluding figures) with the command **latex manual**. The resulting device independent file *manual.dvi* may then be converted to *PostScript* or previewed on screen through a number of widely available routines.

report This directory contains the *LaTeX* source code for the final report outlining the technical details of the *ADPAC08* codes. If *LaTeX* is installed on your system, it is possible to reproduce the final report (excluding figures) with the command **latex report**. The resulting device independent file *finalreport.dvi* may then be converted to *PostScript* or previewed on screen through a number of widely available routines.

src This directory contains all the FORTRAN source code for the *ADPAC08* programs including *SETUP*, *ROTGRID*, *MAKEADGRID*, and *AGTPLT-LCL*.

It is now possible to compile the *ADPAC08* code by issuing the commands

```
cd src/adpac

make
```

On a Cray, the command **make cray** is appropriate, while on an IBM workstation **make aix** is appropriate. Other compilation options are available by typing *make help*. The compilation of the executable module for *ADPAC08* will require roughly 20 megabytes of disk space.

A.4 Running the Distribution Demonstration Test Cases

Once the *make* facility has properly completed compiling the *ADPAC08* source code, it is possible to run the test cases provided with the standard distribution. It is recommended that the sample cases be tested to verify proper compilation and extraction of the *ADPAC08* distribution.

In order to run the demonstration cases, it is necessary to begin in the *demo* directory. From the *ADPAC08* source code directory, the *demo* directory may be entered by issuing the command

cd ../../demo

Several test cases are provided with the standard distribution to illustrate the operation of the code for many different applications. The commands needed to run any demo are similar, so only the case listed under the directory **nasa** will be explained in detail here.

After entering the demo directory, an **ls** command will indicate that the following subdirectories (and possibly others) are available:

```
    nasa/
```

These subdirectories contain the ducted fan demonstration case described above, as well as a sample case (at least partially) for each of the 10 standard configurations described in Chapter 5. To run the multiple blade row ducted fan demonstration case, enter the *nasa* subdirectory by issuing the command **cd nasa**. Now, the **ls** command reveals:

```
  nasa.input          nasa.boundata       nasa.mesh
  nasa.output.save    nasa.converge.save
```

The **nasa** directory contains the data to run a test case for the NASA 1.15 pressure ratio ducted fan. This geometry is representative of a 25:1 bypass ratio turbofan engine fan, and has been tested extensively both experimentally and numerically. This test case employs two blade rows (a rotor and a stator) and the multiple blade rows are treated using the circumferential averaging technique described in Section 2.2. The mesh corresponds to Standard Configuration #10, and the mesh and appropriate mesh indices are illustrated in Figure A.1. The multiple-block mesh for this test case is contained in *nasa.mesh*, and may be viewed using the *PLOT3D* program. The flow Mach number is 0.75, and the calculation is performed at 100% design speed (9167 rpm). For the purposes of this demonstration, an inviscid calculation using 3 levels of multigrid has been configured.

The next step in the solution process is to simply run the *ADPAC08* program for this case. The standard input file **nasa.input** and the boundary data file **nasa.boundata** are provided to run the program (these files are listed in this manual as sample files in Sections 3.6 and 3.7). The steady flow solution is generated by issuing the command

```
../../src/adpac08/adpac <nasa.input >nasa.output
```

The computation time required to generate the steady state solution may take up to four hours on a workstation-class computer. Once the steady flow solution has been generated, the **ls** command will reveal the following files:

```
nasa.restart.new       nasa.p3drel        nasa.p3dabs
nasa.converge          nasa.input         nasa.output
nasa.converge.save     nasa.output.save
```

The file *nasa.restart.new* contains the restart file necessary to continue this run from the point of termination. The files *nasa.p3dabs* and *nasa.p3drel* contain the absolute and relative flow *PLOT3D* flow variable information, respectively. The file *nasa.output* is the new standard output file, and should be compared with the file *nasa.output.save* to verify that the program has performed the calculation correctly. It may be of interest to examine these steady flow results with *PLOT3D* at this point (see Ref. [11] for details).

A plot of the convergence history for this case is given in Figure A.2. The "jumps" in the residual history are a result of the "full" multigrid startup procedure, and should not be considered inappropriate.

The standard output file **nasa.output** should be compared with the listing provided in Section 3.10 to make sure that the code has performed the calculation properly.

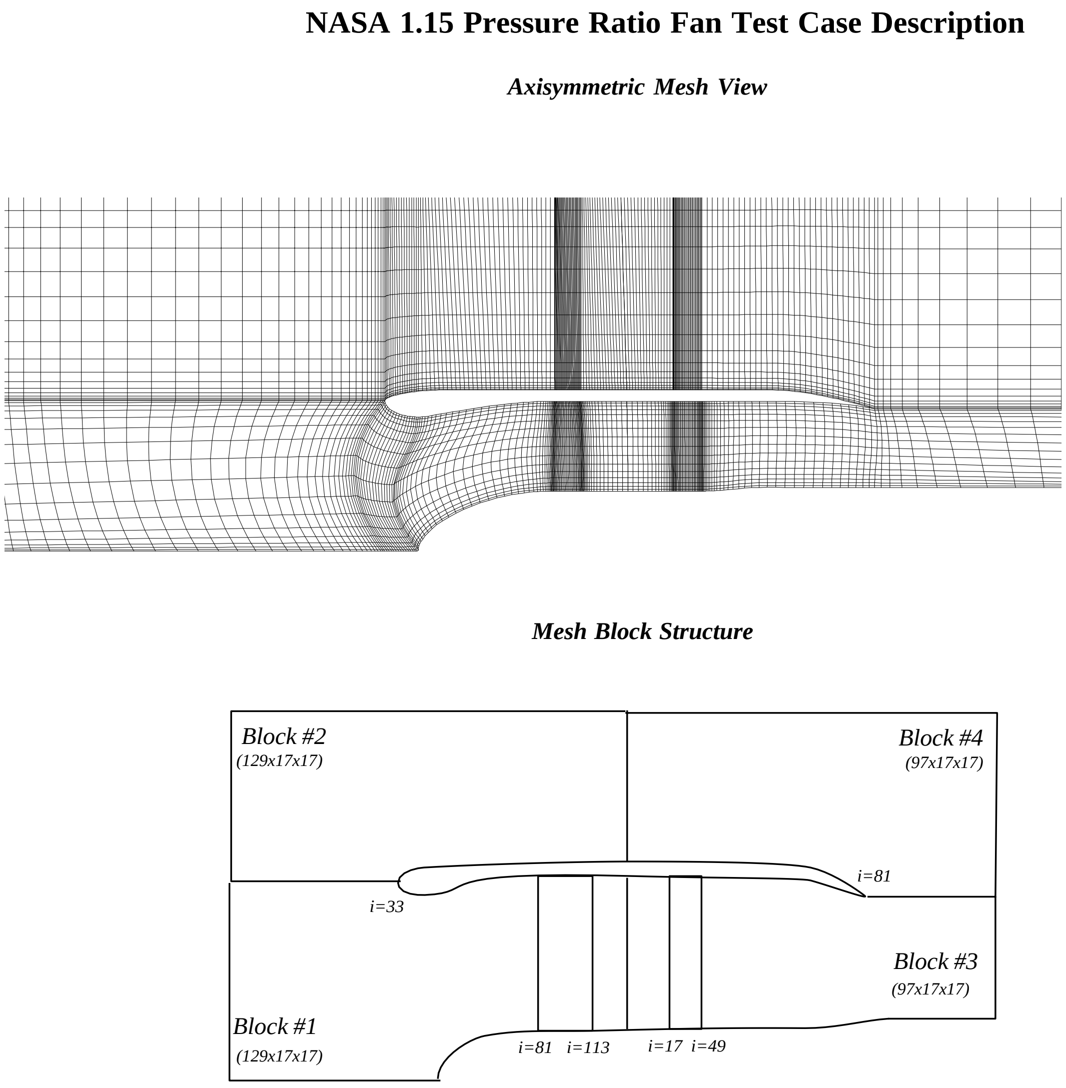


Figure A.1: NASA 1.15 Pressure Ratio Fan Test Case

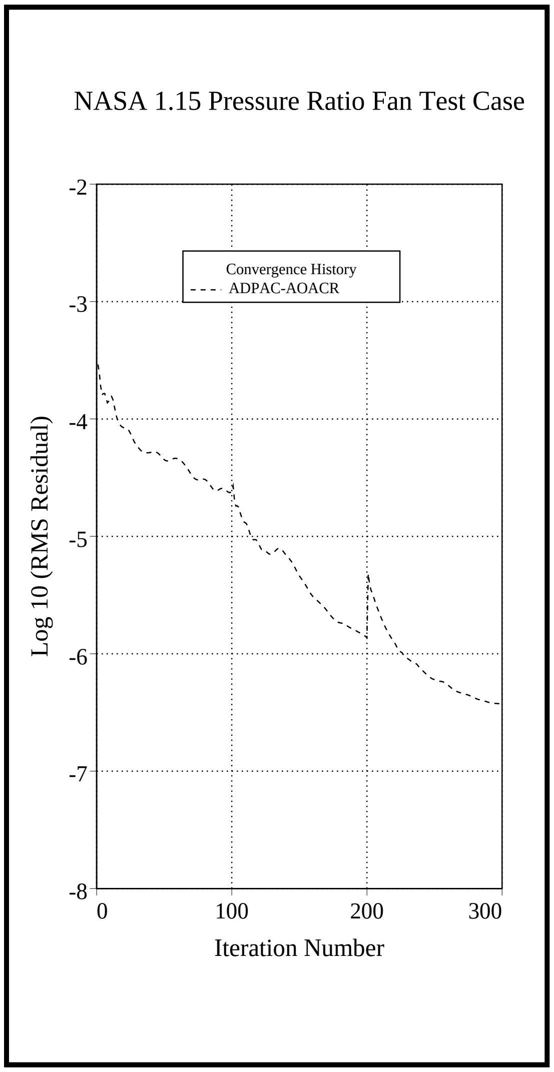


Figure A.2: *ADPAC08* Convergence History for NASA 1.15 Pressure Ratio Fan Test Case

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY (*Leave blank*)	**2. REPORT DATE** September 1994	**3. REPORT TYPE AND DATES COVERED** Final Contractor Report

4. TITLE AND SUBTITLE
Investigation of Advanced Counterrotation Blade Configuration Concepts for High Speed Turboprop Systems
Task 8–Cooling Flow/Heat Transfer Analysis
User's Manual

5. FUNDING NUMBERS
WU–538–03–11
C–NAS3–25270

6. AUTHOR(S)
E.J. Hall, D.A. Topp, N.J. Heidegger, and R.A. Delaney

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Allison Engine Company
P.O. Box 420
Indianapolis, Indiana 46206–0420

8. PERFORMING ORGANIZATION REPORT NUMBER
E–9025

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Lewis Research Center
Cleveland, Ohio 44135 – 3191

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
NASA CR–195360

11. SUPPLEMENTARY NOTES
Project Manager, Christopher J. Miller, Propulsion Systems Division, organization code 2770, (216) 433–6179.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified - Unlimited
Subject Categories 07 and 34

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 words)***

The focus of this task was to validate the ADPAC code for heat transfer calculations. To accomplish this goal, the ADPAC code was modified to allow for a Cartesian coordinate system capability and to add boundary conditions to handle spanwise periodicity and transpiration boundaries. This User's Manual describes how to use the ADPAC code as developed in Task 5, NAS3–25270, and including the modifications made to date in Tasks 7 and 8, NAS3–25270.

14. SUBJECT TERMS
Turbomachinery; Fan; Duct; Navier-Stokes; Multiple block; ADPAC

15. NUMBER OF PAGES 362

16. PRICE CODE A16

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT**